

Please tick ✓ or click if using MS WORD

 FOUNDATION DIPLOMA DEGREE MASTER

Assignment Coversheet

Please complete all details required clearly. For softcopy submissions, please ensure this cover sheet is included at the start of your document or in the file folder.

Assignment & Course Details:

Subject Code: (e.g. XCAT1234) XBFP_3026N	Subject Name (e.g. Fundamentals of Computing): Final Year Project (Part 2)		
Course (e.g. Bachelor in Computing) : Bachelor of Information System (Hons) Enterprise Information Systems			
Lecturer Name: Ms Siti Fazilah Binti Shamsudin			
Assessment Due Date: (dd/mm/yy)	28-7-2024	Assessment Title:	Development of Supply Chain Management System Using Blockchain technology for Customization Furniture Industry

I/We declare that:

- This assignment is my/our own original work, except where I/we have appropriately cited the original source.
- This assignment or parts of it has not previously been submitted for assessment in this or any other subject.
- I/We allow the assessor of this assignment to test any work submitted by me/us, using text comparison software for plagiarism.
(For more information, Please read the Academic Integrity Guidelines)

Name : Goo Han Cong Student ID: 0133677 Email : 0133677@student.uow.edu.my Mobile No: 011-18668016 Signature: <i>Goo</i> Date: 28 - 07 - 2024	Name : Student ID: Email : Mobile No: Signature: Date:	Name : Student ID: Email : Mobile No: Signature: Date:
Name : Student ID: Email : Mobile No: Signature: Date:	Name : Student ID: Email : Mobile No: Signature: Date:	Name : Student ID: Email : Mobile No: Signature: Date:

For office use only – Lecturer comments (if applicable)	Marks Breakdown



**UOW
MALAYSIA**

PART OF THE UNIVERSITY
OF WOLLONGONG AUSTRALIA
GLOBAL NETWORK

**Development of Supply Chain Management System Using Blockchain Technology for
Customization Furniture Industry**

Presented to

School of Computing and Creative Media University of Wollongong Malaysia

By

Goo Han Cong

0133677

Table of Contents

List of Tables.....	12
List of Figures.....	15
Chapter 1 Introduction.....	22
1.1 Research Background	22
1.2 List of Problems.....	23
1.3 Problem Statement.....	23
1.4 Problem Solution	23
1.5 Project Questions	24
1.6 Project Objectives	24
1.7 Project Scope	25
1.8 Significance of Project.....	27
1.9 Conceptual model of Blockchain Supply Chain Management System.....	28
Chapter 2 Literature Review	32
2.1 Knowledge Domain	32
2.1.1 Supply Chain Management.....	32
2.1.1.1 Relationship between Supply Chain Management and Customer Satisfaction	32
2.1.1.2 Relationship between Supply Chain Management and Customer Satisfaction in SME-scale Customization Furniture Industry	33
2.1.2 Business Process of Customization Furniture Factory.....	34
2.1.3 Supply Chain Management System	35
2.1.4 Advanced Technologies in Supply Chain Management System	36
2.1.4.1 Blockchain	36
2.1.4.2 IOT (Internet of Things).....	37
2.1.4.3 Artificial Intelligence and Machine Learning (AI and ML).....	38
2.1.4.4 Automation and Robotics.....	38
2.1.5 Review of Knowledge Domain.....	39
2.2 Technical Domain	40
2.2.1 Blockchain	40
2.2.1.2 Smart Contract	41
2.2.1.3 Blockchain platform.....	41
2.2.1.3.1 Ethereum	41
2.2.1.3.2 IBM Blockchain.....	42
2.2.1.3.3 Hyperledger Fabric	42
2.2.1.4 Methodology for Choosing Blockchain Platform	43
2.2.2 Programming Languages for Web-Based and Blockchain System.....	44
2.2.2.1 HTML	44

2.2.2.2 CSS3	44
2.2.2.3 JavaScript.....	44
2.2.2.4 Python	45
2.2.2.5 Java	45
2.2.2.5 PHP	46
2.2.2.6 MySQL	46
2.2.2.7 GoLang	47
2.2.2.8 Solidity.....	47
2.2.2.9 Vyper	47
2.2.3 Tools and Frameworks	49
2.2.3.1 Web3.js.....	49
2.2.3.2 Node.js	49
2.2.3.3 Bootstrap	49
2.2.3.4 Laravel	50
2.2.3.5 Truffle.....	50
2.2.3.6 Hardhat.....	51
2.2.3.7 Ganache.....	51
2.2.3.8 Embedded JavaScript.....	51
2.2.3.9 Express.....	52
2.2.4 Software Development Methodologies.....	53
2.2.4.1 Waterfall model.....	53
2.2.4.2 Agile model.....	54
2.2.4.3 V-model.....	55
2.2.4.4 Spiral model	56
2.2.4.5 Prototype model	57
2.2.5 Research Instrument.....	58
2.2.5.1 Questionnaire	58
2.2.5.2 Observation	58
2.2.5.3 Interview	59
2.2.5.4 Focus Group.....	59
2.2.5.5 Experiment.....	60
2.2.6 Research Design.....	61
2.2.6.1 Qualitative Research	61
2.2.6.2 Quantitative Research	61
2.2.6.3 Mixed-methods research	62
2.2.7 Existing System	63
2.2.7.1 Magaya Supply Chain System	63

2.2.7.2 FreightPOP	63
2.2.7.3 Precoro	63
2.2.7.4 IBM Blockchain Supply Chain Solution.....	63
2.2.7.5 Summarized Table of Existing System	64
2.2.8 Review of Technical Domain.....	66
2.2.8.1 Blockchain	66
2.2.8.2 Smart Contract	66
2.2.8.3 Blockchain Platforms	66
2.2.8.4 Programming Languages, Tools, and Frameworks	66
2.2.8.5 Software Development Methodology	67
2.2.8.6 Future Improvements	67
Chapter 3 Preliminary Study.....	69
3.1 Overview.....	69
3.2 Project Research Design	69
3.2.1 Design Science Research Methodology (DSRM) Table	69
3.2.2 Project Design Table	71
3.2.3 Evaluation Table.....	72
3.3 Operational Framework	73
3.4 Project Research Instruments.....	75
3.5 Data Collection	76
3.5.1 Population and Sample.....	76
3.6 Chosen Software Development Lifecycle.....	77
3.6.1 Proposed Agile Model.....	78
Chapter 4 Analysis	79
4.1 Introduction.....	79
4.2 Questionnaire Details	79
4.2.1 General Information – Demographic	79
4.2.2 Experience on Ordering Process from Supplier and Customer.....	80
4.2.3 Question on Blockchain Supply Chain Management System.....	80
4.3 Results of the Overall Findings.....	81
4.4 Requirements	92
4.4.1 Use Case Diagram.....	92
4.4.2 Use Case Description.....	93
4.4.3 Hardware and Software Requirement	127
4.4.4 Product Features.....	128
4.4.5 Functional Requirements	129
4.4.6 Non-Functional Requirements	132

4.4.6.1 Usability System Requirements	132
4.4.6.2 Reliability System Requirements.....	132
4.4.6.3 Functionality System Requirements	133
4.4.6.4 Maintainability System Requirements	133
4.4.6.5 Efficiency System Requirements	133
4.4.6.6 Portability System Requirements.....	134
Chapter 5 Design.....	135
5.1 Technical Drawings.....	136
5.1.1 Class Diagram.....	136
5.1.2 Entity Relationship Diagram.....	137
5.1.3 Relational Schema.....	138
5.1.4 Sequence Diagram	139
5.1.5 Activity Diagram.....	154
5.1.7 Context Diagram.....	169
5.2 System Framework	170
5.3 Design Principle.....	172
5.3.1 Visibility.....	172
5.3.2 Consistency	172
5.3.3 Constraint.....	172
5.3.4 Feedback	172
5.4 Mock Up	173
Chapter 6 Implementation	182
6.1 Overview of Directory Structure.....	182
6.1.1 Root Level File	183
6.1.2 Directories.....	184
6.2 Overview of README.md.....	185
6.3 Overview of Utility Files Implementation	186
6.3.1 Authentication Middleware.....	186
6.3.2 Database Configuration.....	187
6.3.2.1 Database Connection.....	187
6.3.2.2 Database Initialization.....	187
6.3.2.3 Default Data Insertion.....	188
6.4 Overview of Blockchain Configuration.....	190
6.4.1 Smart Contract Implementation ('contracts')	190
6.4.1.1 Migration.sol.....	190
6.4.1.2 OrderChain.sol	191
6.4.2 Network Settings for Smart Contract ('truffle-config.js')	192

6.4.3 Smart Contract Deployment ('migrations')	192
6.4.4 Blockchain Interaction with WoodChain ('blockchain.js').....	193
6.5 Overview of M-V-C and Router Directories.....	194
6.5.1 Importing Required Modules and Utilities	194
6.5.1.1 Directory 'models'	194
6.5.1.2 Directory 'controllers'.....	194
6.5.1.3 Directory 'routes'	195
6.5.2 Implementation of Views	196
6.6 Overview of index.js.....	198
6.6.1 Importing Required Modules and Libraries.....	198
6.6.2 Setting Up Session Management	199
6.6.3 Setting Up View Engine and Middleware.....	199
6.6.4 Defining Routes	199
6.6.5 Configuring Server.....	199
6.7 Basic-Level Functionalities Implementation	200
6.7.1 Shared Functionality	200
6.7.1.1 F001 Sign Up	200
6.7.1.1.1 Route	200
6.7.1.1.2 Model	201
6.7.1.1.3 Controller	203
6.7.1.1.4 View	204
6.7.1.2 F002 Log In.....	206
6.7.1.2.1 Route	206
6.7.1.2.2 Controller	207
6.7.1.2.3 View	208
6.8 Intermediate-Level Functionalities Implementation	209
6.8.1 Shared Functionality	210
6.8.1.1 F006 Check Order Status	210
6.8.1.1.1 Route	210
6.8.1.1.2 Model	210
6.8.1.1.3 Controller	211
6.8.1.1.4 View	213
6.8.1.2 F007 Check Invoice	215
6.8.1.2.1 Route	215
6.8.1.2.2 Model	215
6.8.1.2.3 Controller	217
6.8.1.2.4 View	218

6.8.1.3 F011 Check Order Details.....	221
6.8.1.3.1 Route	221
6.8.1.3.2 Controller	221
6.8.1.3.3 View	222
6.8.1.4 F013 Upload Profile Photo	224
6.8.1.4.1 Route	224
6.8.1.4.2 Model	224
6.8.1.4.3 Controller	225
6.8.1.4.4 View	227
6.8.1.5 F015 Search Order	229
6.8.1.5.1 Route	229
6.8.1.5.2 Model	229
6.8.1.5.3 Controller	230
6.8.1.5.4 View	231
6.8.1.6 F017 Sign Out	233
6.8.1.6.1 Route	233
6.8.1.6.2 View	233
6.8.2 User Functionality.....	235
6.8.2.1 F003 Select Supplier	235
6.8.2.1.1 Route	235
6.8.2.1.2 Model	235
6.8.2.1.3 Controller	236
6.8.2.1.4 View	237
6.8.2.2 F004 Select Product	238
6.8.2.2.1 Route	238
6.8.2.2.2 Model	238
6.8.2.2.3 Controller	239
6.8.2.2.4 View	240
6.8.2.3 F005 Place Order	243
6.8.2.3.1 Route	243
6.8.2.3.2 Model	243
6.8.2.3.3 Controller	245
6.8.2.3.4 View	247
6.8.2.4 F014 Search Supplier	249
6.8.2.4.1 Route	249
6.8.2.4.2 Model	249
6.8.2.4.3 Controller	250

6.8.2.4.4 View	250
6.8.3 Supplier Functionality	252
6.8.3.1 Confirm Order (F008).....	252
6.8.3.1.1 Route	252
6.8.3.1.2 Model	252
6.8.3.1.3 Controller	253
6.8.3.1.2 View	254
6.8.3.2 F009 Edit Product	257
6.8.3.2.1 Route	257
6.8.3.2.2 Model	258
6.8.3.2.3 Controller	259
6.8.3.2.4 View	262
6.8.3.3 F010 Add Product	266
6.8.3.3.1 Route	266
6.8.3.3.2 Model	266
6.8.3.3.3 Controller	267
6.8.3.3.1 View	267
6.8.3.4 F012 Add Company Description.....	270
6.8.3.4.1 Route	270
6.8.3.4.2 Model	270
6.8.3.4.3 Controller	271
6.8.3.4.4 View	272
6.8.3.5 F016 Search Product	274
6.8.3.5.1 Route	274
6.8.3.5.2 Model	274
6.8.3.5.3 Controller	275
6.8.3.5.4 View	275
6.7 Advanced-Level Functionalities Implementation	277
Chapter 7 Testing	280
7.1 Overview of Software Testing	280
7.2 White Box Testing.....	280
7.3 Black Box Testing	281
7.4 Grey Box Testing	281
7.5 Testing Levels	282
7.5.1 Unit Testing.....	282
7.5.2 Integration Testing	282
7.5.3 System Testing	282

7.6 Test Case	283
7.6.1 Sign Up	283
7.6.2 Log In.....	287
7.6.3 Select Supplier	289
7.6.4 Select Product	290
7.6.5 Place Order.....	292
7.6.6 Check Order Status	294
7.6.7 Check Invoice	295
7.6.8 Confirm Order.....	296
7.6.9 Edit Product	296
7.6.10 Add Product	301
7.6.11 Check Order Details.....	304
7.6.12 Add Company Description.....	304
7.6.13 Upload Profile Photo.....	305
7.6.14 Search Supplier	305
7.6.15 Search Order	306
7.6.16 Search Product	307
7.6.17 Sign Out	308
7.7 User Acceptance Testing	309
7.7.1 UAT 1	309
Chapter 8 Critical Evaluation and Conclusion	310
8.1 Process Evaluation	310
8.3 Challenges.....	311
8.4 Self-Evaluation	311
8.4.1 Enthusiasm for Learning.....	312
8.4.2 Effective Time Management.....	312
8.4.2 Effective Initial Planning	312
8.5 Future Enhancement	313
8.5.1 Consistency	313
8.5.2 System Design	313
8.5.3 Testing	313
8.5.4 User Interface and User Experience.....	313
8.5.5 Security	314
8.5.6 Functionality	314
8.5.7 Blockchain Utilization	315
8.5.8 Application of Machine Learning	315
Appendix A - References.....	316

Appendix B - Gantt Chart.....	326
Appendix C - Turnitin Report	328
Appendix D – Log Sheet	329

List of Tables

Table 1. 1 Project Scope.....	26
Table 2. 1 Knowledge Points of Blockchain Technology	41
Table 2. 2 Summarized Table of Existing System	65
Table 3. 1 Design Science Research Methodology (DSRM)	70
Table 3. 2 Project Design	71
Table 3. 3 System Evaluation Table	72
Table 3. 4 Operational Framework.....	74
Table 3. 5 Different Formats of Questions in Questionnaire	75
Table 3. 6 Population and Sample.....	76
Table 4. 1 Demographic Details.....	79
Table 4. 2 Experience on Ordering Process from Supplier and Customer.....	80
Table 4. 3 Question on Blockchain Supply Chain Management System.....	81
Table 4. 4 Use Case Table – Sign Up.....	94
Table 4. 5 Use Case Table – Log In	96
Table 4. 6 Use Case Table – Select Supplier.....	97
Table 4. 7 Use Case Table – Select Product.....	99
Table 4. 8 Use Case Table – Place Order	101
Table 4. 9 Use Case Table – Check Order Status.....	102
Table 4. 10 Use Case Table – Check Invoice	104
Table 4. 11 Use Case Table – Confirm Order	107
Table 4. 12 Use Case Table – Edit Product.....	109
Table 4. 13 Use Case Table – Add Product.....	112
Table 4. 14 Use Case Table – Check Order Details	114
Table 4. 15 Use Case Table – Add Company Description	115
Table 4. 16 Use Case Table – Upload Profile Photo	118
Table 4. 17 Use Case Table – Search Supplier.....	119
Table 4. 18 Use Case Table – Search Order	121
Table 4. 19 Use Case Table – Search Product.....	123
Table 4. 20 Use Case Table – Sign Out.....	125
Table 4. 21 Hardware Requirement	127
Table 4. 22 Software Requirement.....	127
Table 4. 23 Product Features.....	128
Table 4. 24 REQ_F001.....	129
Table 4. 25 REQ_F002.....	129
Table 4. 26 REQ_F003.....	129

Table 4. 27 REQ_F004.....	129
Table 4. 28 REQ_F005.....	129
Table 4. 29 REQ_F006.....	129
Table 4. 30 REQ_F007.....	130
Table 4. 31 REQ_F008.....	130
Table 4. 32 REQ_F009.....	130
Table 4. 33 REQ_F010.....	130
Table 4. 34 REQ_F011.....	130
Table 4. 35 REQ_F012.....	131
Table 4. 36 REQ_F013.....	131
Table 4. 37 REQ_F014.....	131
Table 4. 38 REQ_F015.....	131
Table 4. 39 REQ_F016.....	131
Table 4. 40 REQ_F017.....	131
Table 4. 41 Non-Functional Requirements – Usability.....	132
Table 4. 42 Non-Functional Requirements – Reliability	132
Table 4. 43 Non-Functional Requirements – Functionality	133
Table 4. 44 Non-Functional Requirements – Maintainability.....	133
Table 4. 45 Non-Functional Requirements – Efficiency.....	133
Table 4. 46 Non-Functional Requirements – Portability	134
Table 7. 1 Test Case – Validate User Type	283
Table 7. 2 Test Case – Validate Company Addres.....	284
Table 7. 3 Test Case – Validate Company Name	285
Table 7. 4 Test Case – Validate Email.....	286
Table 7. 5 Test Case – Validate Password	287
Table 7. 6 Test Case – Validate Email.....	288
Table 7. 7 Test Case – Validate Password	288
Table 7. 8 Test Case – Validate Log In Function.....	289
Table 7. 9 Test Case – Access Control Test	289
Table 7. 10 Test Case – Validate Display of All Registered Suppliers.....	290
Table 7. 11 Test Case – Validate User Capability to Select Suppliers and Display Associated Products	290
Table 7. 12 Test Case – Access Control Test.....	291
Table 7. 13 Test Case – Validate Select Product and Order Summary.....	291
Table 7. 14 Test Case – Validate Proceed to Checkout Function	292
Table 7. 15 Test Case – Validate Access Control	292
Table 7. 16 Test Case – Validate Delivery Date	293

Table 7. 17 Test Case – Validate Place Order Function	294
Table 7. 18 Test Case – Access Control Test.....	294
Table 7. 19 Test Case – Validate Display of All Orders	294
Table 7. 20 Test Case – Access Control Test.....	295
Table 7. 21 Test Case – Validate Check Invoice Function	296
Table 7. 22 Test Case – Validate Confirm Order Function	296
Table 7. 23 Test Case – Access Control Test.....	297
Table 7. 24 Test Case – Validate Display of All Associated Products.....	297
Table 7. 25 Test Case – Validate Product Name.....	298
Table 7. 26 Test Case – Validate Product Description	298
Table 7. 27 Test Case – Validate Product Price	299
Table 7. 28 Test Case – Validate Product Photo.....	300
Table 7. 29 Test Case – Validate Edit Product Information Function	300
Table 7. 30 Test Case – Validate Add Product Function	301
Table 7. 31 Test Case – Validate Product Description	302
Table 7. 32 Test Case – Validate Product Price	303
Table 7. 33 Test Case – Validate Add Product Information Function	304
Table 7. 34 Test Case – Validate Check Order Details Function.....	304
Table 7. 35 Test Case – Validate Company Description	305
Table 7. 36 Test Case – Validate Profile Photo	305
Table 7. 37 Test Case – Validate Supplier Search by Name.....	306
Table 7. 38 Test Case – Validate Order Search by Name.....	307
Table 7. 39 Test Case – Validate Product Search by Name.....	307
Table 7. 40 Test Case – Validate Sign Out Function	308
Table 7. 41 User Acceptance Testing	309

List of Figures

Figure 1. 1 Conceptual Model of SCMS.....	28
Figure 1. 2 Blockchain Supply Chain Management System.....	29
Figure 1. 3 Blockchain Infrastructure	30
Figure 1. 4 Blockchain Process.....	31
Figure 2. 1 Waterfall model (Adobe Communication Team, 2022).....	53
Figure 2. 2 Agile Model (JavaTPoint, 2023)	54
Figure 2. 3 V-Model (TRY QA, n.d.).....	55
Figure 2. 4 Spiral Model (Martin, 2023).....	56
Figure 2. 5 Prototype Model (Infinity and Board Infinity, 2023)	57
Figure 3. 1 Proposed Agile Model	78
Figure 4. 1 Company Name	81
Figure 4. 2 Role in Industry	81
Figure 4. 3 Role in Company	82
Figure 4. 4 Year in Industry	82
Figure 4. 5 Company Size.....	83
Figure 4. 6 Frequency of Placing Order.....	83
Figure 4. 7 Way to Manage Ordering Process	84
Figure 4. 8 Challenges in Managing Ordering Process.....	84
Figure 4. 9 Frequency of Mistake Occurring	85
Figure 4. 10 Effectiveness of SCMS.....	85
Figure 4. 11 Level of Ordering Problem Impact Production	86
Figure 4. 12 Factors of Using SCMS	86
Figure 4. 13 Productivity Improvement of Using Blockchain SCMS	87
Figure 4. 14 Awareness of Blockchain Technology	87
Figure 4. 15 Level of Trust of Using Blockchain SCMS	88
Figure 4. 16 Willingness of Using Blockchain SCMS.....	88
Figure 4. 17 Correlation of Role in Company and Level of Awareness of Blockchain SCMS	89
Figure 4. 18 Correlation of Role in Company and Level of Trust of Blockchain SCMS	90
Figure 4. 19 Correlation of Role in Company and Level of Willingness of Using Blockchain SCMS	91
Figure 4. 20 Use Case Diagram – Blockchain SCMS	92
Figure 4. 21 State Chart Diagram – Sign Up	94
Figure 4. 22 State Chart Diagram – Log In.....	96
Figure 4. 23 State Chart Diagram – Select Supplier	97
Figure 4. 24 State Chart Diagram – Select Product	99
Figure 4. 25 State Chart Diagram – Place Order	101

Figure 4. 26 State Chart Diagram – Check Order Status	103
Figure 4. 27 State Chart Diagram – Check Invoice	105
Figure 4. 28 State Chart Diagram – Confirm Order.....	107
Figure 4. 29 State Chart Diagram – Edit Product	110
Figure 4. 30 State Chart Diagram – Add Product	112
Figure 4. 31 State Chart Diagram – Check Order Details.....	114
Figure 4. 32 State Chart Diagram – Add Company Description.....	116
Figure 4. 33 State Chart Diagram – Upload Profile Photo	118
Figure 4. 34 State Chart Diagram – Search Supplier.....	120
Figure 4. 35 State Chart Diagram – Search Order	122
Figure 4. 36 State Chart Diagram – Search Product	124
Figure 4. 37 State Chart Diagram – Sign Out	126
Figure 5. 1 Class Diagram	136
Figure 5. 2 Entity Relationship Diagram	137
Figure 5. 3 Relational Schema.....	138
Figure 5. 4 Sequence Diagram – Sign Up.....	139
Figure 5. 5 Sequence Diagram – Log In	139
Figure 5. 6 Sequence Diagram – Select Supplier.....	140
Figure 5. 7 Sequence Diagram – Select Product.....	141
Figure 5. 8 Sequence Diagram – Place Order.....	142
Figure 5. 9 Sequence Diagram – Check Order Status.....	143
Figure 5. 10 Sequence Diagram – Check Invoice.....	144
Figure 5. 11 Sequence Diagram – Confirm Order	145
Figure 5. 12 Sequence Diagram – Edit Product.....	146
Figure 5. 13 Sequence Diagram – Add Product.....	147
Figure 5. 14 Sequence Diagram – Check Order Details	148
Figure 5. 15 Sequence Diagram – Add Company Description	149
Figure 5. 16 Sequence Diagram – Upload Profile Photo	150
Figure 5. 17 Sequence Diagram – Search Supplier.....	151
Figure 5. 18 Sequence Diagram – Search Order.....	151
Figure 5. 19 Sequence Diagram – Search Product.....	152
Figure 5. 20 Sequence Diagram – Sign Out.....	153
Figure 5. 21 Activity Diagram – Sign	154
Figure 5. 22 Activity Diagram – Log In.....	155
Figure 5. 23 Activity Diagram – Select Supplier.....	155
Figure 5. 24 Activity Diagram – Select Product	156
Figure 5. 25 Activity Diagram – Place Order	157

Figure 5. 26 Activity Diagram – Check Order Status	158
Figure 5. 27 Activity Diagram – Check Inovice	159
Figure 5. 28 Activity Diagram – Confirm Order	160
Figure 5. 29 Activity Diagram – Edit Product	161
Figure 5. 30 Activity Diagram – Add Product	162
Figure 5. 31 Activity Diagram – Check Order Details.....	163
Figure 5. 32 Activity Diagram – Add Company Description.....	164
Figure 5. 33 Activity Diagram – Upload Profile Photo	165
Figure 5. 34 Activity Diagram – Search Supplier.....	166
Figure 5. 35 Activity Diagram – Search Order	166
Figure 5. 36 Activity Diagram – Search Product	167
Figure 5. 37 Sign Out.....	168
Figure 5. 38 Context Diagram – Blockchain SCMS.....	169
Figure 5. 39 Model View Controller Architecture Representation	170
Figure 5. 40 Layered Architecture Diagram	171
Figure 5. 41 Sign Up Page	173
Figure 5. 42 Log In Page.....	173
Figure 5. 43 Select Supplier Page (User Homepage).....	174
Figure 5. 44 Select Product Page	174
Figure 5. 45 Place Order Page	175
Figure 5. 46 User Manage Order Page.....	175
Figure 5. 47 User Side Order Details Modal	176
Figure 5. 48 Manage Product Page (Supplier Homepage).....	176
Figure 5. 49 Manage Product Modal	177
Figure 5. 50 Add Product Modal.....	177
Figure 5. 51 Supplier Manage Order Page.....	178
Figure 5. 52 Supplier Side Order Details Modal	178
Figure 5. 53 Sample PDF Invoice.....	179
Figure 5. 54 User Profile Options	179
Figure 5. 55 Supplier Profile Options Modal.....	180
Figure 5. 56 Upload Profile Photo Modal.....	180
Figure 5. 57 Add Company Description Modal	181
Figure 5. 58 Sign Out Confirmation Modal.....	181
Figure 6. 1 Directory Structure before Executing System	182
Figure 6. 2 Directory Structure after Executing System	183
Figure 6. 3 Overview of ‘README.md’	185
Figure 6. 4 Overview of ‘authMiddleware.js’	186

Figure 6. 5 Code Snippet in ‘supplierRoutes.js’	186
Figure 6. 6 Database Connection in ‘database.js’	187
Figure 6. 7 Database Initialization in ‘database.js’	187
Figure 6. 8 Default User Insertion in ‘database.js’	188
Figure 6. 9 Default Supplier Insertion in ‘database.js’	188
Figure 6. 10 Overview of ‘Migration.sol’	190
Figure 6. 11 Overview of ‘OrderChain.sol’	191
Figure 6. 12 Code Snippet of ‘truffle-config.js’	192
Figure 6. 13 Overview of ‘1_initial_migration.js’	192
Figure 6. 14 Overview of ‘2_deploy_order_chain.js’	193
Figure 6. 15 Initialization and Setup Part in ‘blockchain.js’	193
Figure 6. 16 Modules and Utilities Importing in ‘mainModels.js’	194
Figure 6. 17 Modules and Utilities Importing in ‘supplierModels.js’	194
Figure 6. 18 Modules and Utilities Importing in ‘userModels.js’	194
Figure 6. 19 Modules and Utilities Importing in ‘mainControllers.js’	194
Figure 6. 20 Modules and Utilities Importing in ‘supplierControllers.js’	194
Figure 6. 21 Modules and Utilities Importing in ‘userControllers.js’	195
Figure 6. 22 Modules and Utilities Importing in ‘mainRoutes.js’	195
Figure 6. 23 Modules and Utilities Importing in ‘supplierRoutes.js’	195
Figure 6. 24 Modules and Utilities Importing in ‘userRoutes.js’	195
Figure 6. 25 Overview of ‘views’ Directory Structure	196
Figure 6. 26 Overview of ‘index.js’	198
Figure 6. 27 Basic-Level Functional Requirements.....	200
Figure 6. 28 Route for Main Page Rendering in ‘main.js’	200
Figure 6. 29 Route for Sign Up Request Handling in ‘main.js’	201
Figure 6. 30 Model for User Confidential Validation in ‘mainModels.js’	201
Figure 6. 31 Model for New User Creation in ‘mainModels.js’	201
Figure 6. 32 Model for New User Creation in ‘mainModels.js’	202
Figure 6. 33 Controller for Signing Up in ‘mainController.js’	203
Figure 6. 34 Displaying Error Message in ‘main.ejs’	204
Figure 6. 35 Utilization of Bootstrap Framework in ‘main.ejs’	204
Figure 6. 36 Required Fill in ‘main.ejs’	205
Figure 6. 37 Login Section in ‘main.ejs’	205
Figure 6. 38 Route for Login Page Rendering in ‘main.js’	206
Figure 6. 39 Route for Login Request Handling in ‘main.js’	206
Figure 6. 40 Route for Login Page Message Handling in ‘main.js’.....	206
Figure 6. 41 Controller for Logging In in ‘mainController.js’	207

Figure 6. 42 Sign-Up Section in ‘main.ejs’	208
Figure 6. 43 Intermediate-Level Functional Requirements	209
Figure 6. 44 Route for User Manage Order Page Rendering in ‘userRoutes.js’	210
Figure 6. 45 Route for Supplier Manage Order Page Rendering in ‘supplierRoutes.js’	210
Figure 6. 46 Model for Retrieving Order in ‘userModels.js’	210
Figure 6. 47 Model for Retrieving Order in ‘supplierModels.js’	211
Figure 6. 48 Controller for Displaying User Manage Order Page in ‘userController.js’	211
Figure 6. 49 Controller for Displaying Supplier Manage Order in ‘supplierController.js’	212
Figure 6. 50 Code Snippet in ‘manageorder.ejs’	213
Figure 6. 51 Overview of ‘supplierorder.ejs’	214
Figure 6. 52 Route for Invoice Rendering in ‘main.js’	215
Figure 6. 53 Model for Retrieving Order Details in ‘mainModels.js’	215
Figure 6. 54 Model for Retrieving Order’s Product Details in ‘mainModels.js’	216
Figure 6. 55 Controller for Displaying Invoice in ‘mainController.js’	217
Figure 6. 56 Overview of ‘orderdetails.ejs’	218
Figure 6. 57 Overlay Message for Invoice Generation in ‘manageorder.ejs’ and ‘supplierorder.ejs’	219
Figure 6. 58 Client-Side JavaScript in ‘manageorder.ejs’	219
Figure 6. 59 Client-Side JavaScript in ‘supplierorder.ejs’	220
Figure 6. 60 Route for Unconfirmed Order Detail Rendering in ‘main.js’	221
Figure 6. 61 Controller for Displaying Unconfirmed Order Details in ‘mainController.js’	221
Figure 6. 62 Unconfirmed Order Modal in ‘manageorder.ejs’	222
Figure 6. 63 Client-Side JavaScript in ‘userorder.ejs’	223
Figure 6. 64 Client-Side JavaScript in ‘supplierorder.ejs’	223
Figure 6. 65 Route for Upload Profile Photo Request Handling in ‘main.js’	224
Figure 6. 66 Controller for Uploading Profile Photo in ‘mainController.js’	224
Figure 6. 67 Setting Up Mutler for Profile Photo Upload in ‘mainController.js’	225
Figure 6. 68 Controller for Uploading Profile Photo in ‘mainController.js’	226
Figure 6. 69 Profile Photo Upload Modal in ‘uploadprofilemodal.ejs’	227
Figure 6. 70 Client-Side JavaScript ‘uploadprofilemodal.ejs’	228
Figure 6. 71 Route for User Manage Order Page Rendering in ‘userRoutes.js’	229
Figure 6. 72 Route for Supplier Manage Order Page Rendering in ‘supplierRoutes.js’	229
Figure 6. 73 Model for Searching Order in ‘userModels.js’	229
Figure 6. 74 Controller for Searching Order in ‘userController.js’	230
Figure 6. 75 Controller for Searching Order in ‘supplierController.js’	231
Figure 6. 76 Search Order Section in ‘manageorder.ejs’	231
Figure 6. 77 Search Order Section in ‘supplierorder.ejs’	232
Figure 6. 78 Client-Side JavaScript in ‘userorder.ejs’	232

Figure 6. 79 Route for Sign Out Request Handling in ‘main.js’	233
Figure 6. 80 Sign Out Confirmation Modal in ‘signoutmodal.ejs’	233
Figure 6. 81 Client-Side JavaScript in ‘signoutmodal.ejs’	234
Figure 6. 82 Route for User Homepage Rendering in ‘userRoutes.js’	235
Figure 6. 83 Model for Retrieving Supplier Information in ‘userRoutes.js’	235
Figure 6. 84 Controller for Displaying User Homepage in ‘userRoutes.js’	236
Figure 6. 85 Supplier List Section in ‘userhome.ejs’	237
Figure 6. 86 Route for Select Product Page Rendering in ‘userRoutes.js’	238
Figure 6. 87 Model for Retrieving Supplier Details in ‘userModels.js’	238
Figure 6. 88 Model for Retrieving Products by Supplier ID in ‘userModels.js’	239
Figure 6. 89 Controller for Displaying Select Product Page in ‘userController.js’	239
Figure 6. 90 Overview of Select Product Page	240
Figure 6. 91 Order Summary Section in ‘selectproduct.ejs’	241
Figure 6. 92 Overview of Client-Side JavaScript in ‘selectproduct.ejs’	241
Figure 6. 93 Function ‘proceedToCheckout()’ in ‘selectproduct.ejs’	242
Figure 6. 94 Route for Place Order Page Rendering in ‘userRoutes.js’	243
Figure 6. 95 Model for Fetching Product Details in ‘userModels.js’	243
Figure 6. 96 Model for Creating Order in ‘userModels.js’	244
Figure 6. 97 Model for Creating Order Detail in ‘userModels.js’	244
Figure 6. 98 Controller for Rendering Place Order Page in ‘userController.js’	245
Figure 6. 99 Controller for Recording Order in ‘userController.js’	246
Figure 6. 100 Overview of ‘placeorder.ejs’	247
Figure 6. 101 Client-Side JavaScript in ‘placeorder.ejs’	248
Figure 6. 102 Route for Search Supplier Request Handling in ‘userRoutes.js’	249
Figure 6. 103 Model for Searching Supplier in ‘userModels.js’	249
Figure 6. 104 Controller for Searching Supplier in ‘userController.js’	250
Figure 6. 105 Search Supplier Bar in ‘userhome.ejs’	250
Figure 6. 106 Supplier List Section in ‘userhome.ejs’	251
Figure 6. 107 Route for Update Order Status Request Handling in ‘supplierRoutes.js’	252
Figure 6. 108 Model for Updating Order Status in ‘supplierModels.js’	252
Figure 6. 109 Controller for Updating Order Status in ‘supplierController.js’	253
Figure 6. 110 Order Table Section in ‘supplierorder.ejs’	254
Figure 6. 111 Confirm Order Confirmation Modal in ‘supplierorder.ejs’	255
Figure 6. 112 Client-Side JavaScript in ‘supplierorder.ejs’	256
Figure 6. 113 Route for Supplier Homepage Rendering in ‘supplierRoutes.js’	257
Figure 6. 114 Route for Updating Product Information Request Handling in ‘supplierRoutes.js’	257
Figure 6. 115 Route for Deleting Product Request Handling in ‘supplierRoutes.js’	257

Figure 6. 116 Model for Retrieving Product in ‘supplierModels.js’	258
Figure 6. 117 Model for Adding Product in ‘supplierModels.js’	258
Figure 6. 118 Model for Deleting Product in ‘supplierModels.js’	259
Figure 6. 119 Controller for Displaying Supplier Homepage in ‘supplierController.js’	259
Figure 6. 120 Setting Up Mutler for Product Photo Upload in ‘supplierController.js’	260
Figure 6. 121 Controller for Updating Product Information in ‘supplierController.js’	261
Figure 6. 122Controller for Deleting Product in ‘supplierController.js’	261
Figure 6. 123 Overview of ‘supplierhome.ejs’	262
Figure 6. 124 Manage Product Modal in ‘manageproductmodal.ejs’	263
Figure 6. 125 Confirmation Modals in ‘manageproductmodal.ejs’	264
Figure 6. 126 Client-Side JavaScript in ‘manageproductmodal.ejs’	265
Figure 6. 127 Route for Adding Product Request Handling in ‘supplierRoutes.js’	266
Figure 6. 128 Model for Adding Product in ‘supplierModels.js’	266
Figure 6. 129 Controller for Adding Product in ‘supplierController.js’	267
Figure 6. 130 Add Product Section in ‘supplierhome.ejs’	267
Figure 6. 131 Add Product and Confirmation Modal in ‘suppliermodals.ejs’	268
Figure 6. 132 Client-Side JavaScript in ‘suppliermodals.ejs’	269
Figure 6. 133 Route for Update Supplier Description Request Handling in ‘supplierRoutes.js’	270
Figure 6. 134 Model for Retrieving Supplier Description in ‘supplierModels.js’	270
Figure 6. 135 Model for Updating Supplier Description in ‘supplierModels.js’	270
Figure 6. 136 Controller for Updating Supplier Description in ‘supplierController.js’	271
Figure 6. 137 Add Company Description Modal in ‘suppliermodals.ejs’	272
Figure 6. 138 Client-Side JavaScript in ‘suppliermodals.ejs’	273
Figure 6. 139 Route for Search Supplier Request Handling in ‘supplierRoutes.js’	274
Figure 6. 140 Model for Retrieving Product Name in ‘supplierModels.js’	274
Figure 6. 141 Controller for Searching Product Name in ‘supplierController.js’	275
Figure 6. 142 Search Product Bar in ‘supplierhome.ejs’	275
Figure 6. 143 Product List Section in ‘supplierhome.ejs’	276
Figure 6. 144 Advanced-Level Functional Requirements.....	277
Figure 6. 145 Exported Functions in ‘blockchain.js’	277
Figure 6. 146 Function for Placing Order in ‘blockchain.js’	278
Figure 6. 147 Function for Updating Order Status in ‘blockchain.js’	279
Figure 6. 148 Functions for Fetching Orders and Order Details in ‘blockchain.js’	279
Figure 7. 1 Types of Software Testing (EduBridge, 2023)	280

Chapter 1 Introduction

1.1 Research Background

It's been a long time that the Small and Medium-sized Enterprises (SMEs) and microenterprises play an important role in the business. As Bayraktar and Mulan stated, they normally represent the lifeblood of various industries since they form most firms in most countries which is 95% or more (2019). They also account for 90% of the business population, 60%-70% of employment, and 55% of GDP in developed economies. In Malaysia, SMEs make up 97.2% of all business establishments and generate 38.2% for Malaysia (Organisation for Economic Co-operation and Development, 2022). However, the SMEs are the businesses who are facing most of the challenges. These challenges are composed of intense competition, organizational internal and external communications, economic and financial, human resources, and supply chains problem. The mentioned problems have become serious since the revolution of industry 4.0. It is due to the limited ability of SMEs and microenterprises in terms of adapting to digitalization, cooperation and various changes that have happened in industry 4.0 (Bak & Reicher, 2023).

Meanwhile, the customization furniture manufacturers might face most of the supply chain problems that have been mentioned above and it is directly affecting their daily operational efficiency, product quality, and customer satisfaction (Samani, 2023).

In a study conducted by Červený et al. in 2022, the opportunities and challenges of Industry 4.0 for the furniture industry were examined. The research findings recommend that small and medium-sized enterprises (SMEs) should forge collaborative relationships for both production and development, enabling SMEs to actively participate in supplier-buyer networks. The study also underscores the significance of transitioning from traditional business models to Industry 4.0, as it promises swift improvements. However, it's worth noting that the participants in this research collectively agree that the current state of the furniture industry predominantly reflects Industry 2.0 practices.

One prevailing theme echoed through the research: the critical need for a Supply Chain Management System (SCMS) fine-tuned to the specific requisites of SMEs in this sector. Surprisingly, existing research has not adequately delved into this critical aspect, highlighting a notable gap in understanding and solutions within the existing literature.

To bridge this gap, this research endeavors to develop an intuitive and tailored SCMS with a focus on web-based and blockchain technologies. By leveraging the capabilities of the web and blockchain, the proposed SCMS aims to optimize supply chain processes, minimize inefficiencies, and foster growth for SMEs in the customization furniture manufacturing domain. This forward-looking approach not only aligns with the recommendations for Industry 4.0 but also positions SMEs to thrive in an increasingly digitized and interconnected business landscape.

1.2 List of Problems

- i. Customization furniture means more options and more complexity in terms of managing the varied raw material, effective coordination and communication pose challenges, resulting in additional time and costs (FarEye. 2023).
- ii. Customized furniture manufacturers which are in SME-scale normally lack the ability in terms of limited capital and knowledge to do the digital transformation, including the supply chain process (Bak & Reicher, 2023).
- iii. Fluctuations in market demand for a specific raw material could lead to insufficient inventory management and product demand estimation (Abuzaid et al., 2023).

1.3 Problem Statement

Nowadays, most of the customization furniture manufacturers are facing supply chain management problems, especially the entities who are categorized as SMEs or microbusinesses. Despite the related businesses deeply understand the problem is existing but they are still not able to address the root cause of the problem as they have agreed on the current state of furniture companies is still at the level of Industry 2.0 and they are still facing the same supply chain problems (Červený et al., 2022). These problems have negatively affected the of the companies' profitability, reputation, and efficiency. The possible causes of the problem are significantly related to the knowledge of the workers, the business scale, capital, and the availability of suitable SCMS that will support the digital transformation of SMEs and microbusiness in the customization furniture industry. Perhaps a study of a Supply Chain Management System for the customization furniture industry would help the industry to streamline and resilient supply chain.

1.4 Problem Solution

An easy-to-use with the simple structure of Supply Chain Management System (SCMS) will be helpful in terms of addressing the identified problems. This kind of system will take the available cost and knowledge of the related businesses into account, so SCMS will be able to reach in this field and make it affordable for the customized furniture manufacturers which are categorized in SMEs and microbusinesses. Meanwhile, the system shall be designed with the well understanding of embeddedness theory so the features of system can be more in line with the needs and structures of the relevant fields. By using this system, the manufacturer will streamline the process of order raw materials. The blockchain integration adds an extra layer of security, making transactions more secure and resistant to tampering. It means that blockchain acts as notary in this system.

This perfectly addresses the troublesome of repeatedly order process due to communication problems with upstream suppliers and customers. The system also provides a fundamental for further data utilization, so the fluctuation of demands and customer behaviours will have a chance to be manageable and predictable. This will greatly improve the operational performance in supply chain management.

1.5 Project Questions

- i. What are the existing technologies and methods used in supply chain management, particularly for SMEs and microbusinesses?
- ii. How does the Blockchain SCMS manage the flow and movement of goods to enhance the supply chain efficiency?
- iii. Will the adoption of a tailored SCMS enhance the operational efficiency of customization furniture industry?

1.6 Project Objectives

- i. To study existing technologies and methods used in Supply Chain Management System.
- ii. To design and develop a Blockchain Supply Chain Management System on the customization furniture industry.
- iii. To validate the efficiency and effectiveness of the Blockchain Supply Chain Management System on the customization furniture industry.

1.7 Project Scope

Project Scope		
Level	Function	Description
Basic	<ul style="list-style-type: none"> The system shall be able to let users register and log in an account. The Supply Chain Management System shall have the ability of showing the raw materials, orders and users' information. 	<ul style="list-style-type: none"> The system will allow the users to register and log in their account. The system will allow the users to gain the specific raw materials and users' information that they want.
Intermediate	<ul style="list-style-type: none"> The Supply Chain Management System shall have the ability of editing the raw materials and users' information. The Supply Chain Management System shall have the ability of searching suppliers, products, and orders by using search bar. The Supply Chain Management System shall have the ability of updating order status. The Supply Chain Management System shall have the ability of placing order. The Supply Chain Management System shall have the ability of invoice generation. 	<ul style="list-style-type: none"> The system will allow the users to edit raw materials and users' information that they want. The system will allow the users to search the suppliers, products, and orders that they want. The system will allow the users to update the order status. The system will allow the users to place the order for different suppliers. The system will allow the users to receive invoice that is generated automatically.

Advanced	<ul style="list-style-type: none"> The Supply Chain Management System shall have the integration with the blockchain. 	<ul style="list-style-type: none"> The system will allow the user to record the order and order status to blockchain thus increasing the security of transaction.
-----------------	--	--

Table 1. 1 Project Scope

1.8 Significance of Project

The benefit from this study is not only addressing the current problems. It also brings the target audience and society move towards and approach to the direction of Society 5.0 and industry 5.0.

i. SMEs and Microenterprises in Customized Furniture Manufacturing (Industry 4.0):

The project brings significant benefits for the SMEs and microenterprises in customized furniture manufacturing and even the whole SMEs in furniture industry. It addresses the problem with those manufacturers who are stuck in Industry 2.0. The simple yet useful blockchain-based SCMS can empowers businesses to streamline their operations hence enhance productivity and profitability. It meets the principle of Industry 4.0 by embracing digital transformation (Ilanković et al., 2019).

ii. Government and Regulatory Bodies (Society 5.0):

It supports the larger social change envisioned in Society 5.0 by encouraging the integration of digital solutions within SMEs in the furniture manufacturing industry. It motivates governing bodies and governments to promote and ease the adoption of digital technologies, which is crucial for societal well-being, economic growth, and sustainability.

iii. Society and Consumers (Society 4.0 / 5.0):

A well-managed supply chain means more efficient and effective supply chain. This also refers to the efficient resource use, and reducing waste, for example, the traffic oil as well as the exhaust fumes that are needed in the delivery process. The most important, the timeliness to respond to customers' demands also will be optimized. Finally, it benefits both manufacturers and customers since the on-time delivery are always being enabled. This also represents the phenomenon of alignment between human and Information System (IS), it means that the Society 5.0 will be approached more.

1.9 Conceptual model of Blockchain Supply Chain Management System

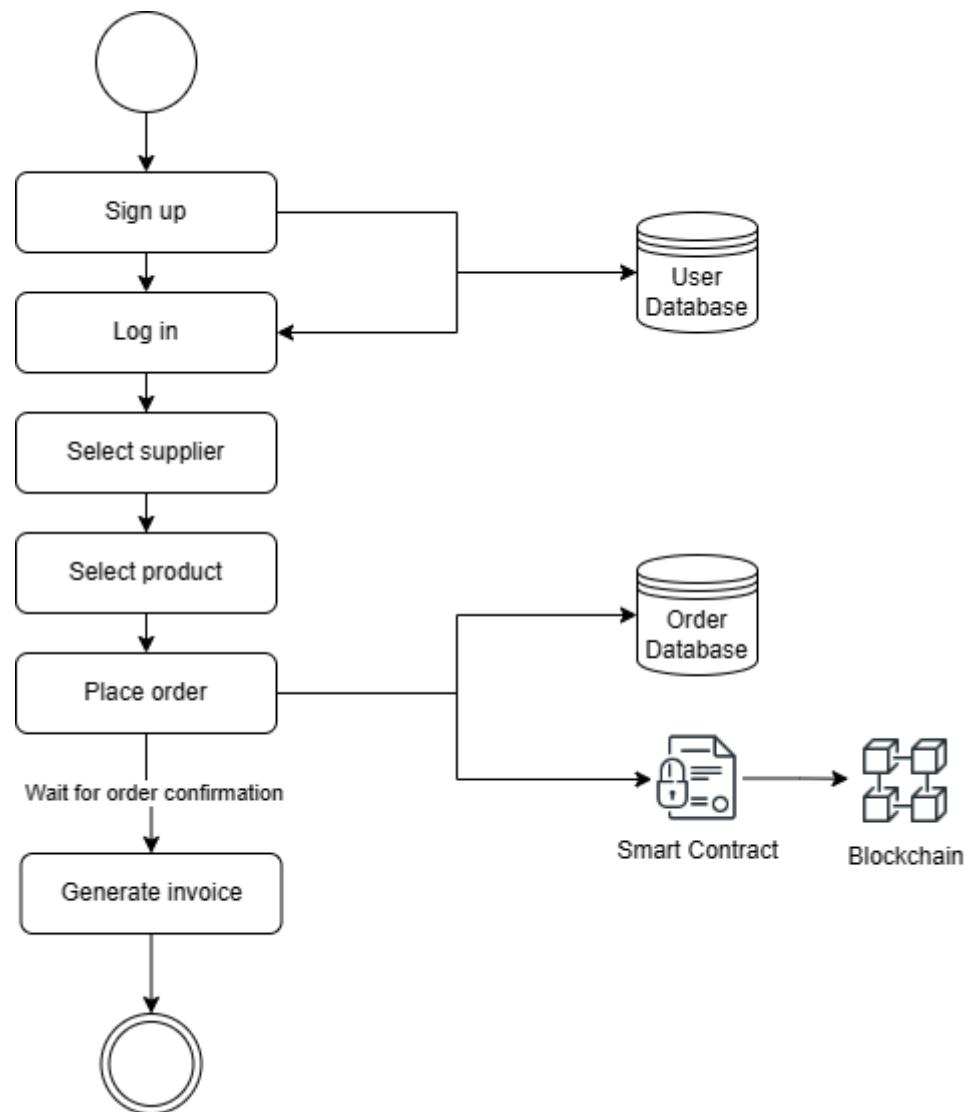


Figure 1. 1 Conceptual Model of SCMS

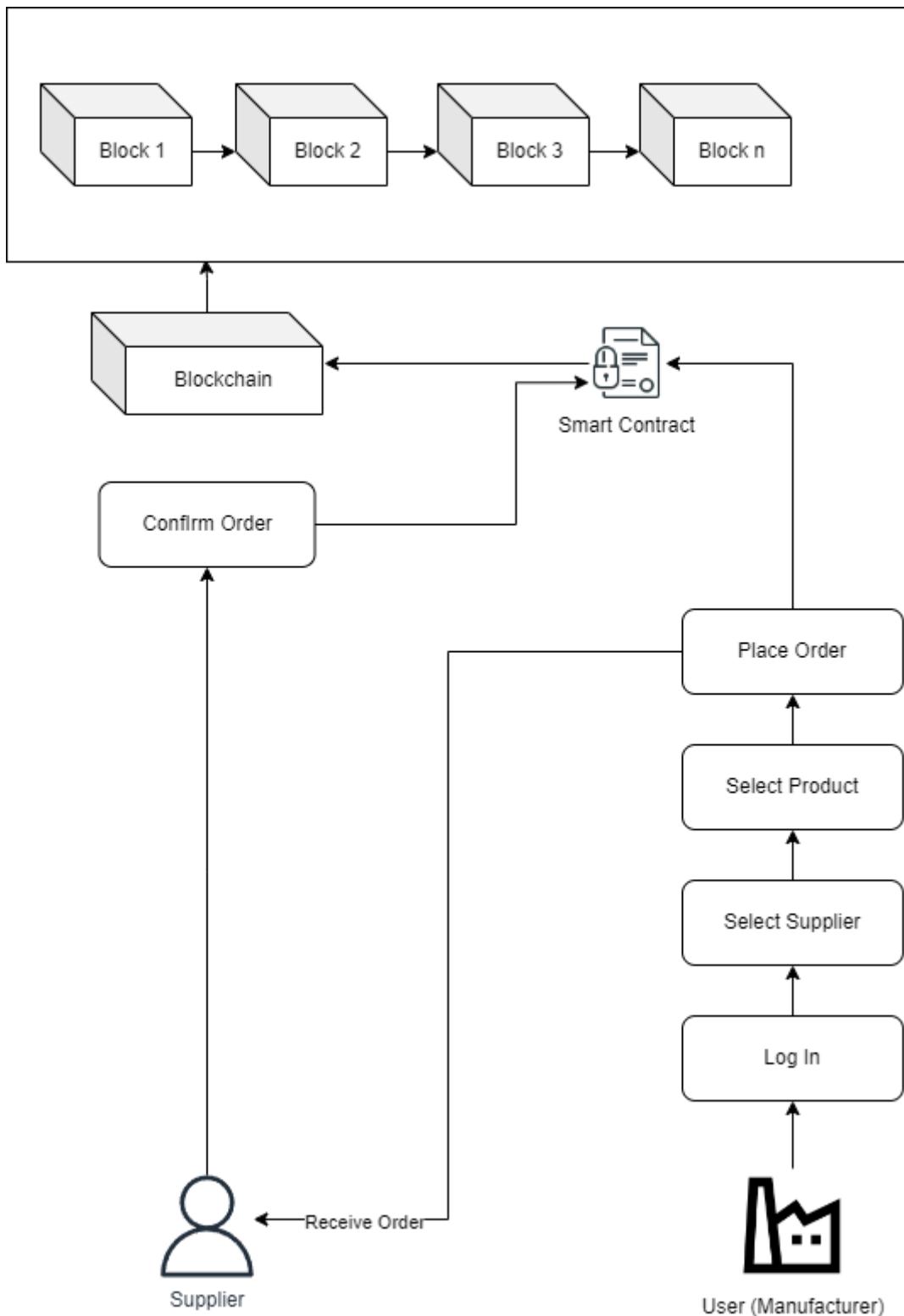


Figure 1. 2 Blockchain Supply Chain Management System

Distributed Ledger

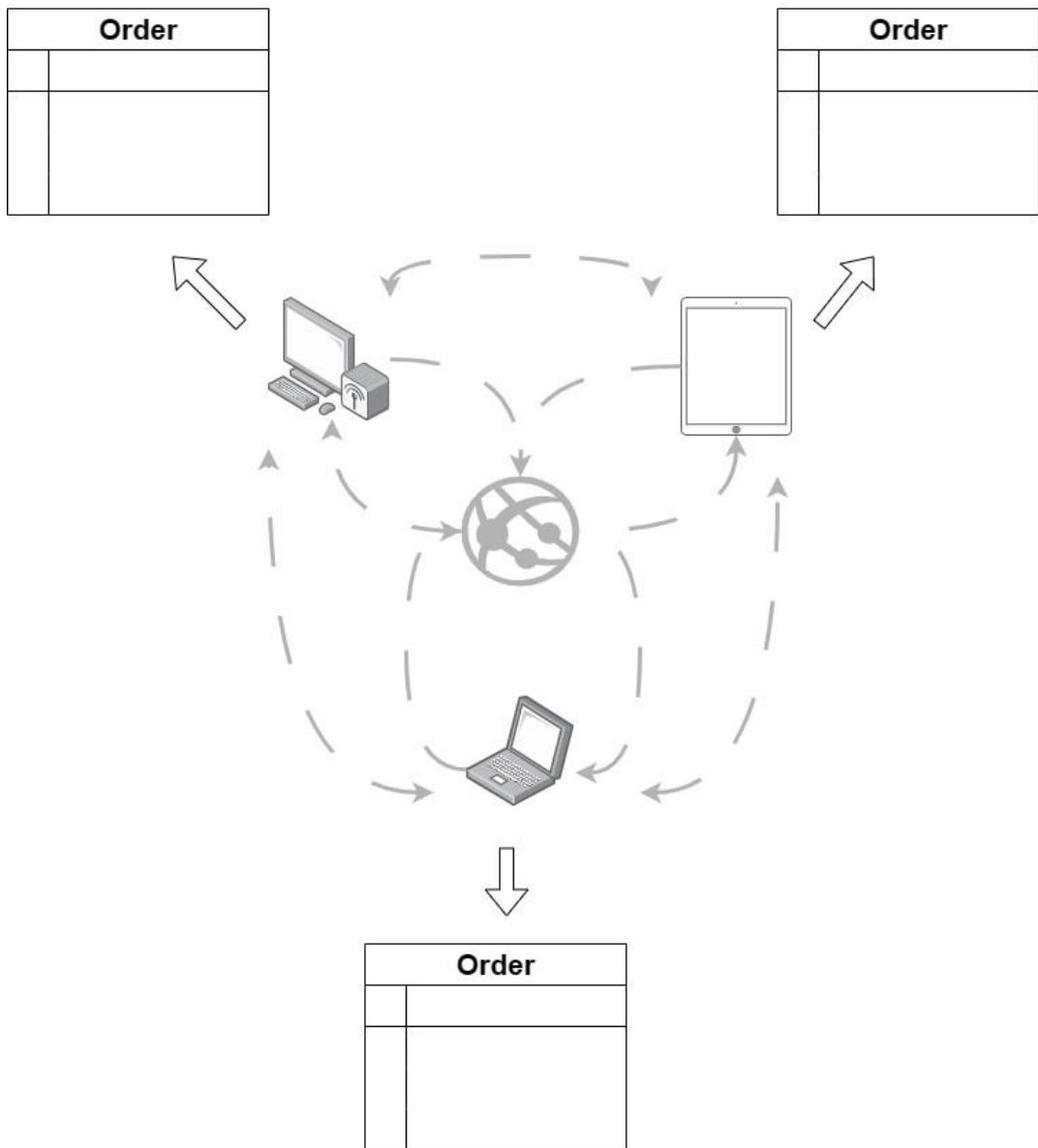


Figure 1. 3 Blockchain Infrastructure



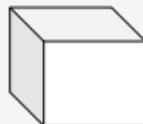
A network participant forwards a transaction proposal to the blockchain system



A secured block is formulated



Transaction verified and approved by other participants (nodes)



New blockchain added

+



All participants acquire latest block

Figure 1. 4 Blockchain Process

Chapter 2 Literature Review

As previously highlighted, this research centers on the creation of a Supply Chain Management System (SCMS) tailored for the Small and Medium-sized Enterprises (SMEs) operating in the customization furniture industry. The forthcoming literature review is poised to make substantial contributions by delving into key knowledge domains, including supply chain management principles, associated business processes, and the prevailing technologies employed in contemporary supply chain management systems.

From a technical standpoint, this review will not only investigate the alternatives in system development methodologies but also explore the diverse array of research instruments and programming languages that are pertinent to the development of an effective SCMS. By comprehensively examining these aspects, we aim to lay a robust foundation for the subsequent phases of our research, ensuring a well-informed and strategic approach to the development of a tailored SCMS for SMEs in the customization furniture sector.

Finally, it helps to examine the level of necessity of the development of supply chain management system in this industry.

2.1 Knowledge Domain

2.1.1 Supply Chain Management

Muheesi has defined supply chain management as a comprehensive system of managing information, materials, and services flow from the raw material suppliers to the end customer (2022). It means that this process involves the design, planning, execution, and monitoring of all activities from the procurement of raw materials to the delivery of the final product to the end customer.

Normally, business will pursue a more efficient supply chain management process to increase their profitability, flexibility, and agility (Irfan et al., 2019). This statement also represents the final goals and objectives of supply chain management for business in most of the time. It directly determines the ability of a business to create and capture the value.

2.1.1.1 Relationship between Supply Chain Management and Customer Satisfaction

In certain industries, the impact of raw material supply chain disruptions on customer satisfaction may vary considerably. Take, for instance, a bakery producer that supplies its products nationwide. If a disruption in raw material supply prevents the production of certain items, it may not necessarily have a significant effect on customer satisfaction. The reasoning behind this lies in the diversified product range of the bakery — not all products depend on the same set of raw materials. Therefore, if specific items are temporarily unavailable, customers still have alternative choices, potentially mitigating the impact on their satisfaction. Additionally, the broad geographical scope of the business might allow it to absorb disruptions in specific regions without causing a substantial overall decline in customer

satisfaction. Of course, effective communication about the supply chain challenges and any adjustments made can play a crucial role in managing customer expectations and minimizing the impact on satisfaction levels.

Based on information gathered from various researchers, including Kanike (2023) and Katsaliaki et al. (2020), several crucial elements in the realm of supply chain disruption that significantly impact customer satisfaction can be identified. They are industry sensitivity to raw material disruption, product diversity and substitution, customer expectations and tolerance, and geographical scope.

2.1.1.2 Relationship between Supply Chain Management and Customer Satisfaction in SME-scale Customization Furniture Industry

In the context of the SME-scale customization furniture industry, supply chain disruptions wield a notable impact on both customer satisfaction and business operations. Unlike larger industries, the intricate nature of crafting customized furniture entails a higher degree of dependence on specific raw materials. Consequently, any disruption in the supply chain can significantly affect the production of tailored furniture pieces. The limited product diversity in this niche, where each item is uniquely crafted, amplifies the challenges posed by raw material shortages. Customers seeking bespoke furniture solutions may find fewer alternative options available, heightening the potential negative impact on their satisfaction.

Moreover, the bespoke nature of the industry implies that customer expectations are often finely tuned, and their tolerance for disruptions or substitutions may be lower compared to industries with more standardized products. The geographical scale of SMEs in this sector may not offer the same buffering effect as larger enterprises, as their customer base is often more localized. In such a scenario, disruptions in specific regions can have a more direct and pronounced effect on overall customer satisfaction. Clear communication and transparency become imperative in managing customer expectations, as any hiccup in the supply chain can reverberate not only through customer satisfaction levels but also through the smooth functioning of the business operations.

2.1.2 Business Process of Customization Furniture Factory

MTO stands for make-to-order, which is a production approach and strategy where products are produced when the order is received (Upadhyay et al., 2023). The typical nature of this manufacturing approach is the order is made based on the specific requirements from customer order rather than being produced in anticipation of future demand.

According to my research findings, there is a notable gap in academic literature regarding the supply chain management process within the SME-scale customization furniture industry. However, several websites explicitly classify the SME-scale customization furniture sector as a prime illustration of the Make to Order (MTO) model, such as ACTouch (2023) and Due (2023). While scholarly discussions on this industry's supply chain intricacies are currently lacking, the online sources offer valuable insights by categorizing and characterizing it as a noteworthy example of Make to Order manufacturing.

These websites clearly described the business process's attributes of the enterprise which apply the MTO manufacturing strategy. Among them, ACTouch clearly gave the details of SME-scale customization furniture industry's business process (2023). The results of the study and the real-life situation is roughly the same, that is, the custom furniture factory will not have any inventory of raw materials. They will start purchasing raw material based on customer demand before start making the furniture. The following are the steps of SME-scale custom furniture industry business process in terms of the order fulfilment and supply chain management.

- Step 1: Negotiate with clients and agents.
- Step 2: Determine the raw materials that clients-side wants.
- Step 3: Order the raw materials from various suppliers based on the prices and availability of the products.
- Step 4: Receive the raw materials and start to produce the final product.
- Step 5: Deliver and install the customized furniture.

It's important to highlight that steps 2 to 4 are consistently prone to supply chain challenges. Instances such as employees forgetting to order specific raw materials or clients requesting changes to the chosen raw materials after the supplier order is placed are common issues encountered in this phase. These challenges underscore the critical nature of effective supply chain management and communication during these steps to mitigate disruptions and ensure a smooth supply chain process.

2.1.3 Supply Chain Management System

SCMS, which is also known and Supply Chain Management System, acts as software to manage the supply chain. It is a software-based solution designed to streamline and optimize the processes involved in the supply chain. Therefore, the system normally encompasses a wide range of components to be helpful for the users in the daily supply chain management activities.

Based on the previous research, the components in this system may include but not limited to:

- Inventory management
- Procurement
- Order processing
- Demand Forecasting
- Logistics and distribution
- Supplier Relationships Management (SRM)
- Integration with Other System

They are all essential elements in the process of supply chain management. However, it is crucial to acknowledge that, in the context of the SME-scale customization furniture industry, a nuanced approach may be necessary. While all these components are theoretically integral, the practicality and cost-effectiveness of implementing each one must be carefully considered. In this specific industry, where resources may be constrained, prioritizing key features that directly address pressing supply chain challenges becomes paramount. Notably, focusing on streamlined procurement and efficient order processing, which are identified as immediate needs in the industry, could provide a more targeted and cost-effective solution to promptly address supply chain issues.

While specific citations supporting this industry-specific approach may be limited, the rationale is rooted in the understanding that tailored solutions should align with the industry's unique characteristics and challenges, emphasizing agility and responsiveness over a comprehensive but potentially resource-intensive implementation of all supply chain components.

2.1.4 Advanced Technologies in Supply Chain Management System

In the rapidly evolving landscape of supply chain management, advancements in technology have become integral to enhancing efficiency, transparency, and agility. From leveraging artificial intelligence and machine learning for predictive analytics to embracing the Internet of Things (IoT) for real-time tracking, the adoption of these cutting-edge technologies is reshaping the traditional paradigms of supply chain operations. This section delves into the forefront of technological innovations driving the evolution of modern SCMS. The technologies like IoT, blockchain, AI, ML, and robotics will be reviewed in this section.

2.1.4.1 Blockchain

Blockchain is an innovative technology that is a decentralized and distributed digital ledger technology used for recording the transaction. Song et al. have directly defined it as a “record-keeping system that will stores information about the transaction records shared peer-to-peer across all computers, devices within its network” (2019).

This author also mentioned the advantages of the blockchain’s application in supply chain management. These benefits can be succinctly outlined as follows:

- Decentralization and Trust

Operating on a decentralized network, blockchain reduces reliance on intermediaries. This decentralized nature ensures that transactions are transparent, secure, and immutable, thereby fostering trust among suppliers.

- Enhanced Security

Blockchain employs cryptographic techniques to secure data, effectively preventing unauthorized access, fraud, and tampering. This inherent security enhances the resilience of supply chain data against cyber threats.

- Improved Traceability

Every transaction is meticulously recorded in a block, forming an unbroken chain. This cohesive structure allows organizations to seamlessly trace the origin, movement, and handling of products throughout the supply chain.

- Efficiency and Cost Reduction

Automated processes play a pivotal role in diminishing paperwork and minimizing manual errors. Smart contracts, programmed to execute predefined actions autonomously, further contribute to this efficiency. The cumulative effect is an overall enhancement in supply chain efficiency, resulting in significant cost savings.

- Real-time Visibility

Blockchain furnishes real-time updates regarding the status of products, allowing participants to closely track inventory levels, monitor shipments, and stay informed about delivery times.

This heightened visibility significantly augments decision-making capabilities and responsiveness within the supply chain.

2.1.4.2 IOT (*Internet of Things*)

IoT stands for Internet of Things, it is a technology to integrate the physical devices and sensors to gather and transmit data. It is one of the important technologies in terms of digitalization. In the realm of supply chain management, IoT has been widely used across the entire supply chain. Ben-Daya et al. pointed out that the IoT includes 4 main essential layers (2017). They are sensing layer, networking layer, service layer, and interface layer. These layers generally represent the structure of IoT.

De Vass et al.'s research (2022) underscores a multitude of benefits stemming from the integration of IoT (Internet of Things) into supply chain management. The findings highlight a transformative impact across various facets of the supply chain, elucidating key advantages that contribute to enhanced efficiency and responsiveness. These benefits can be succinctly outlined as follows:

- Transport Sector Advancements

By using the IoT, it enables the track-and-trace systems, fleet controls, vehicle tracking, route optimization, and so on.

- Supplier Integration Enhancements

IoT is a crucial component for enhancing relationships with suppliers. For example, it facilitates information exchange, forecasting, and optimization of operational aspects between retailers and suppliers. Besides the information from the literature, we can know that it plays an important role in the Just-in-Time (JIT) system, thereby increasing switching costs for both retailers and suppliers.

- Customer-Centric Applications

In some industries and business models, the IoT can be used for enhancing customers' store experiences and online interactions.

- Delivery Process Optimization

IoT technologies offer real-time tracking, efficient route planning, and automated alerts, leading to improved customer satisfaction and reduced delivery times.

- Real-time Analytics

Since the entire supply chain becomes visible, IoT will offer the ability of displaying real-time information on the location and status of products. This will finally contribute to the analytics then enhance the decision-making and so on.

2.1.4.3 Artificial Intelligence and Machine Learning (AI and ML)

AI stands for Artificial Intelligence, meanwhile, ML stands for Machine Learning. Both are always being used interchangeably. In fact, ML is one of the subsets of AI, it is defined as the development of algorithms and models that allows systems to learn from the data by itself (Sarker, 2021). At the same time, AI refers to the creation of intelligent agents capable of performing tasks that require human intelligence.

Within the field of supply chain management, AI and ML play the crucial role in optimizing and enhancing such process (Alhasawi et al., 2023). The benefits of AI and ML that had been mentioned by these authors including but not limited to:

- Demand Forecasting

AI and ML algorithms analyze historical data, market trends, and external factors to predict future demand more accurately. This helps in optimizing inventory levels and preventing overstock or stockouts. This benefit may not be perceived by the custom furniture factory at all since they are normally adopting MTO strategy. Therefore, this benefit maybe more accessible by the suppliers.

- Inventory Management

ML algorithms can optimize inventory levels by considering factors like lead times, order frequency, and demand variability. This ensures that the right amount of stock is always maintained.

2.1.4.4 Automation and Robotics

Automation and robotics also have played an important role in revolutionizing the supply chain management. It involves various types of robots such as Automated Material Handling Systems (AMHS), Automated Guided Vehicles (AGVs), robotics in manufacturing and so on. Each of them in charge of different tasks in supply chain management, for example AMHS helps in moving, storing, and retrieving materials within a facility or warehouse. AGVs helps in routing and delivering materials (Evjemo et al., 2020). The common benefits of this kind of robot include reducing human resources, enhancing efficiency and effectiveness of business operations.

2.1.5 Review of Knowledge Domain

The identification of factors with a significant impact on customer satisfaction and business operations in this industry substantiates the indirect but substantial influence it holds. Moreover, a notable discovery in earlier research underscores the imperative for a Supply Chain Management System (SCMS) in this sector. Specifically, it emphasizes the prevalent lack of digitalization within the industry, this also represents the reason why the involvers in this industry normally agree they are in the industry 2.0. This dual revelation emphasizes not only the heightened impact on customer satisfaction and business operations but also the pressing need for digital advancements in the supply chain processes of this industry.

In evaluating advanced technologies for the Supply Chain Management System (SCMS), prioritizing blockchain implementation is recommended due to its potential to enhance transparency and security. While IoT offers valuable functions like real-time tracking and analytics, there is some overlap with blockchain. However, considering the cost-effectiveness, a strategic approach may involve selective adoption based on the industry's unique requirements and challenges. It is focusing on blockchain ensures a targeted and cost-effective solution, aligning with the specific needs of the SME-scale customization furniture industry.

2.2 Technical Domain

2.2.1 Blockchain

As mentioned previously, blockchain is an innovative technology that change the way of how a data and information can be recorded. In 2023, Mukta has given a clear overview blockchain technology. There are few important concepts in blockchain technology. These concepts are organized in the following tables:

<i>Knowledge Points of Blockchain Technology</i>		
No	Knowledge Points	Blockchain Technology
1	Decentralization	Blockchain operates on a network of computers. It means that every node will have a copy of the entire blockchain, rather than relying on a single central authority
2	Block and Chain	In this technology, blocks serve as containers for recording lists of transactions and timestamps. A hash pointer is utilized to connect each block to the previous one. The chain is formed by linking these blocks, with each block incorporating the hash value of its preceding block.
3	Consensus Algorithms	Before a block is added to the blockchain, the network must reach consensus that the transaction is valid. The consensus algorithm such as Proof of Work, Proof of Stake, Proof of Authority and Practical Byzantine Fault Tolerance are the most common used algorithms.
4	Security and Transparency	This is the most important benefits of blockchain technology. As every node store the information in form of block and the blocks are linked with the subsequent blocks, it is difficult to change the information. Thus, it enhances the security and transparency.
5	Types of Blockchain	<ul style="list-style-type: none">• Public blockchain The blockchain that is opened to all users and used for activities such as mining and cryptocurrency exchange.• Private blockchain The blockchain that is operated within closed networks, limiting access to selected members, and is employed in various organizational functions like SCM and digital identity.• Consortium blockchain The blockchain that is governed by a group rather than a single entity. It will allow multiple organizations to act as nodes. For example, banks and government organizations may use this blockchain.

		<ul style="list-style-type: none"> • Hybrid blockchain <p>The blockchain that allow the combination of privacy benefits of private blockchains and the transparency benefits of the public blockchain.</p>
--	--	---

Table 2. 1 Knowledge Points of Blockchain Technology

2.2.1.2 Smart Contract

Smart contract is one of the important reasons that increase the effectiveness and efficiency of business operations by using the blockchain technology. Taherdoost even mentioned this is a significant development in blockchain (2023). It refers to a self-executing contract with the terms of the agreement directly written into code (Parate et al., 2023). In short, smart contracts eliminate the need for intermediaries, such as banks and so on. In other words, this is a type of e-agreement that automatically generated once the transaction is made based on the predefined conditions.

With the combination of blockchain's natures such as transparency and security, smart contracts gradually become important for those who have applied the blockchain. Although, it has not been acknowledged widely in terms of legal perspective, there are still a lot of people proactively embrace this technology. Ene directly described this is a new form of the legal agreement (2020).

2.2.1.3 Blockchain platform

Blockchain platform is the underlying technology and infrastructure that will support the development of blockchain-based applications. It provides the necessary tools, protocols, and environment for creating, deploying, and managing the blockchain-based systems and the components within the blockchain network such as smart contracts and so on (Komalavalli et al., 2020).

Based on the research from Kuo et al., (2019), Polge et al. (2021), Kushwaha et al., (2022), and Lawton (2023) there are several common blockchain platforms are available for people to use. Each of them will have different benefits.

2.2.1.3.1 Ethereum

This is one of the oldest established blockchain platforms and it was introduced in 2013.

- Advantages:
 - Ethereum provides an open-source and public platform, fostering a collaborative development environment.
 - Its support for decentralized applications (DApps) and smart contracts enables a wide range of blockchain-based functionalities.
 - The Ethereum Virtual Machine (EVM) allows nodes to execute programs in any language, enhancing flexibility.
- Disadvantages:

- Ethereum's original Proof-of-Work (PoW) consensus mechanism poses scalability challenges and requires significant computational power.
- The transition to Proof-of-Stake (PoS) and Ethereum 2.0 is a complex process, aiming to address scalability and energy efficiency concerns.
- While Ethereum is versatile, the openness of the platform may lead to potential security vulnerabilities that developers need to manage carefully.

2.2.1.3.2 IBM Blockchain

- Advantages:
 - IBM Blockchain offers enterprise-level solutions, ensuring robust security and privacy features.
 - It provides a modular and flexible framework, suitable for various business applications.
 - Integration with IBM's cloud services enhances accessibility and scalability.
- Disadvantages:
 - IBM Blockchain may involve higher costs compared to some open-source alternatives.
 - It could have a steeper learning curve for developers due to its enterprise-focused features.

2.2.1.3.3 Hyperledger Fabric

- Advantages:
 - Hyperledger Fabric is designed for permissioned blockchains, making it suitable for enterprise-level applications with controlled access.
 - Its support for DApps written in general-purpose programming languages enhances flexibility for developers.
 - Fabric's modular architecture allows plugging in different consensus protocols, offering adaptability based on specific use cases.
 - The absence of an underlying cryptocurrency can be advantageous for enterprises not requiring native tokens.
- Disadvantages:
 - The permissioned nature of Hyperledger Fabric might limit its use for fully decentralized applications.
 - The complexity of configuring and managing a Fabric network can be a challenge for users without extensive blockchain expertise.
 - Integrating different consensus protocols requires careful consideration and may add complexity to the network setup.

2.2.1.4 Methodology for Choosing Blockchain Platform

In 2021, Nanayakkara et al. proposed a methodology for choosing the appropriate blockchain to develop the Enterprise Information System. There are 4 essential steps:

- Identification

The tasks in this phase include the identification and review of available blockchain platforms in the market. The review criteria encompass features, capabilities, and technical requirements of the blockchain platform.

- Selection

This phase involves using multi-criteria decision-making method such as SMART to rank the blockchain platform. The higher rank means more suitable for the requirements of enterprise or individual who wants to use it.

- Evaluation

This phase will involve activities such as reviewing the system architecture, libraries, and tools, as well as testing the platform by developing a sample application to assess its feasibility and usability.

- Validation

In this phase, the methodology's overall process protocol is validated by its application to a real-world case study. The results are then compared with those obtained from other existing methods or platforms.

2.2.2 Programming Languages for Web-Based and Blockchain System

2.2.2.1 HTML

HTML stands for Hypertext Markup Language; it is markup language and serves as a fundamental of website construction (GeeksforGeeks, 2023).

- Advantages:
 - Ease of learning and use
 - Wide browser support
 - Versatility
 - Lightweight and fast to load
 - Compatibility with other technologies
- Disadvantages:
 - Limited interactivity
 - Lack of styling
 - Static structure
 - Security risks

2.2.2.2 CSS3

CSS is known as Cascading Style Sheets; it is a language used for styling a HTML document. In other words, it helps to make the HTML document's appearance become prettier by styling the layout, colors, fonts and so on (Cascading Style Sheets, n.d.).

- Advantages:
 - Easy maintenance
 - Fast loading time
 - Responsive design
 - Consistent styling
 - Separation of content and presentation
- Disadvantages:
 - Browser compatibility
 - Limited layout control
 - Performance impact by the extensive use of CSS

2.2.2.3 JavaScript

JavaScript is a scripting language which is used to create and control the content dynamically on the website. It can be used to update and change both HTML and CSS (Jordana, 2023). To create and control the content, it needs to calculate, manipulate, and validate the data.

In the realm of blockchain, this language could be used for develop the Hyperledger Farbric smart contracts, which is chaincode.

- Advantages:
 - Wide adoption
 - Client-side interactivity
 - Wealth of libraries and frameworks supporting
- Disadvantages:
 - Security concern
 - Longer loading time if application is complex
 - Lack of debugging facility

2.2.2.4 Python

In the case of developing web-based system, Phyton could be used for front-end and back-end development (Coursera, 2023). It is a high-level programming language.

- Advantages:
 - Easy to learn
 - Readability and simplicity of syntax
 - Community support
 - Not limited to web development
- Disadvantages:
 - Speed is slower than Java
 - Database access could be a problem since it lacks the support from robust mechanisms

2.2.2.5 Java

Java is also one of the most popular high-level programming languages in current time. Besides that, Java is a multi-platform, object-oriented, and network-centric programming language. Furthermore, it could be used for the development of game, cloud computing, big data, artificial intelligence, and Internet of Things (Amazon, n.d.). APIs and Java Virtual Machine are both important components in this language. APIs provides the ability of communicating between different software such as databases and so on. JVM provides a platform-independent runtime environment for Java Applications. In other words, it helps the communication among Java platforms, devices, and operating systems.

- Advantages
 - Platform independence
 - Well suited for large-scale web application
 - Rich ecosystems such as numerous frameworks
 - Security

- Object-oriented approach
- Disadvantages
 - Slower execution speed
 - Significant memory space use
 - Inability to build complex user interface
 - Not user-friendly syntax

2.2.2.5 PHP

PHP, Hypertext Preprocessor, is one of the most widely used web development programming languages. It is an open-source server-side scripting language (Chris, 2021a).

- Advantages:
 - Cross-platform
 - Open source
 - Easy to learn
 - Sync with all databases
 - Community support
- Disadvantages:
 - Inadequate error handling
 - Limited support for OOP
 - Slower speed
 - Not suitable for large application

2.2.2.6 MySQL

MySQL is a structured query language that serves as primary language used in relational databases. By using this language, the data could be stored, retrieved, and modified easily.

- Advantages
 - Easy to use
 - Portability
 - Standardization
 - Community support
 - Scalability
- Disadvantages
 - Complexity in scaling horizontally
 - Higher costs
 - Limited support for unstructured data

2.2.2.7 GoLang

Go is also known as GoLang as well. It is a Google-developed open-source programming language (Chris, 2021b). Go is known for its simplicity, efficiency, and strong support for concurrent programming.

- Advantages:
 - Easy to learn
 - Growing community support
 - High performance and suitable for small-scale utilities and large-scale distributed systems
 - Widely used in systems programming, cloud services, web development and so on
- Disadvantages:
 - Less expressiveness compared to other languages
 - Limited framework

2.2.2.8 Solidity

Solidity is an object-oriented programming language that is specifically designed for developing smart contracts on Etheruem. This is also the primary language used on Etheruem. Since, it can be used to create the smart contracts, it is able to integrate with the business logic (Simplilearn, 2023).

- Advantages:
 - Purposely design for smart contracts
 - Compatibility with dApp and EVM (Ethereum Virtual Machine)
 - Community support
 - Security mechanisms
 - Ethereum ecosystem support
- Disadvantages:
 - Highly dependent on Etheruem
 - Evolving standards in terms of meeting changes in industry and regular practices

2.2.2.9 Vyper

Vyper is the contract-oriented language that has been used on Ethereum blockchain platform (Inuwa, 2022). It serves as the alternative to Solidity for developing smart contracts. Moreover, it is a pythonic programming language, so the structure of syntax is always simple, readable, and secure enough.

- Advantages:
 - Readability
 - Better security

- Python familiarity
- Disadvantage:
 - Newer language, less community support
 - Omit some features present in Solidity
 - Smaller ecosystems

2.2.3 Tools and Frameworks

2.2.3.1 Web3.js

Zubair stated that Web3.js is an open-source library of JavaScript (2023). This library provides the ability of interacting with the Ethereum blockchain. It acts as the intermediary between DApps and the Ethereum blockchain, so the developers can develop the smart contracts, retrieve the data within the blockchain and send transactions.

- Advantages:
 - Ethereum integration
 - Smart contracts integration
 - Blockchain data accessibility
- Disadvantages:
 - Documentation challenges
 - Ethereum-specific
 - Security issues

2.2.3.2 Node.js

Node.js is an open-source and server-side runtime environment built on the Chrome V8 JavaScript engine (Khare, 2023). It allows the execution on the server and outside of the web browser. Besides that, the author also mentioned that Node.js is designed to be lightweight and efficient so it could be well suited to scalable applications.

- Advantages:
 - Built with an event-driven and non-blocking I/O model
 - Better scalability
 - Community support
 - Fast execution
- Disadvantages:
 - Callback hell
 - Limited CPU utilization

2.2.3.3 Bootstrap

Bootstrap is a commonly used open-source frontend framework used for designing and developing the webpage (Zola, 2022). The Bootstrap framework is constructed upon three fundamental components: HTML, CSS, and JavaScript. Therefore, a set of pre-designed components, styles, and various plugins will be provided in this framework.

- Advantages:
 - Responsive design

- Consistent
 - Time efficiency
 - Customizable design
 - Community Support
- Disadvantages:
 - Generic look
 - Limited design freedom
 - Bigger file size
 - Dependent JavaScript

2.2.3.4 Laravel

Laravel is an open-source PHP web framework. It is designed as simple, readable, and user-friendly framework. It follows the Model-View-Controller (MVC) pattern and provides a range of built-in tools such as Eloquent, Blade templating, and modular packaging system (Jalli, 2022).

- Advantages
 - Developer-friendly syntax
 - Artisan console
 - Better security
 - Various built-in features such as authentication, authorization, caching, and session management
- Disadvantages
 - Relatively difficult to learn
 - Limited flexibility
 - Comprehensive documentation

2.2.3.5 Truffle

Truffle is development framework for Ethereum blockchain. In this framework, a series of tools for developing, testing, and deploying smart contracts will be provided (Tean, 2023). By using the Truffle, the complex process of blockchain development will be simplified by offering development environment, testing framework, and asset pipeline.

- Advantages:
 - Simplified development
 - Smart contract compilation and migration
 - Wide adoption and well community support
- Disadvantages:
 - Difficult to learn
 - Limited to Ethereum

2.2.3.6 Hardhat

Hardhat is an Ethereum development environment that facilitates the creation, testing, and deployment of smart contracts on the Ethereum blockchain (Howell, 2023). Like Truffle, Hardhat also provides tools and features to streamline the development process of smart contracts and DApps.

- Advantages:
 - Extensive plugin systems
 - Built-in support for TypeScript
 - Advanced testing functionality
 - Hardhat network
- Disadvantages:
 - Difficult to learn
 - Smaller community support
 - Newer than Truffle

2.2.3.7 Ganache

Ganache is a personal blockchain used for Ethereum development (Blockchain Council, 2024). It allows developers to create and test smart contracts and DApps (Decentralized Applications) in a local environment, simulating the Ethereum blockchain.

- Advantages:
 - Local blockchain
 - Fast deployment
 - Customizable
- Disadvantages:
 - Limited to development
 - No real-world interaction

2.2.3.8 Embedded JavaScript

Embedded JavaScript (EJS) is a simple templating language that allows developers to generate HTML markup with plain JavaScript (Olusola, 2024). It is used to embed JavaScript code within HTML templates, enabling the creation of dynamic web pages (Olusola).

- Advantages:
 - Easy to use
 - Logic to templates
 - Partial views
 - Seamless integration with Node.js
- Disadvantages:

- Limited Functionality
- Slower performance in comparison to other templating engines
- Less community support

2.2.3.9 Express

Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications (Sharma, 2024). It simplifies the process of building server-side applications in JavaScript (Sharma).

- Advantages:
 - Middleware support
 - Powerful routing mechanism
 - Easy to use
 - Allowing third-party libraries and middleware
- Disadvantages:
 - Minimal structure
 - Callback hell
 - Require having good understanding of Node.js and JavaScript

2.2.4 Software Development Methodologies

2.2.4.1 Waterfall model

Waterfall model is one of the earliest and most straightforward software development approach. It could be called as waterfall model or waterfall methodology. In this model, the process of system development will be split into several phases (Adobe Communication Team, 2022). These phases include requirement gathering, system design and analysis, system implementation, system testing, system deployment, and system maintenance.

In this approach, the linear-sequential life cycle model is well represented as the stages may begin only after the preceding process has been completed. The following figure is cited from Adobe Communication Team (2022) and it demonstrates the steps in this model graphically.

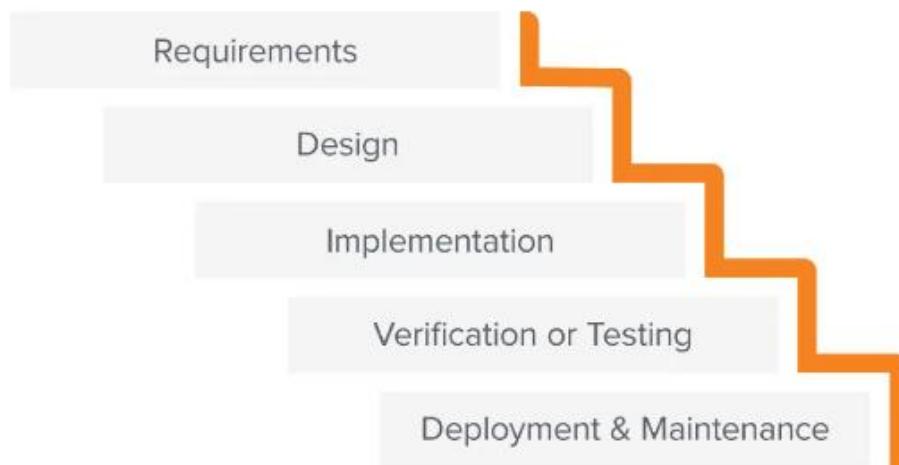


Figure 2. 1 Waterfall model (Adobe Communication Team, 2022)

- Advantages:
 - Simple to understand
 - Clear project objectives
 - Easy to manage
 - Well documented
- Disadvantages:
 - Inflexible to change
 - Late product delivery
 - Limited customer involvement
 - Lower customer satisfaction
 - Costly to change

2.2.4.2 Agile model

In SDLC, agile methodology provides a more rapid and flexible approach to developing system. According to the information from JavaTPoint website, this is an iterative and incremental approach to develop the small or functional increments in every iteration within a short time frames (2023). There are several methods in agile methodology could be chose such as Scrum, XP, Crystal and so on.



Figure 2. 2 Agile Model (JavaTPoint, 2023)

- Advantages:
 - Flexible and adaptable
 - Better customer satisfaction
 - Rapid delivery of value
 - Cost-efficiency
 - Increased collaboration among team members
- Disadvantages:
 - Resource intensive
 - Higher potential for scope creep
 - Lack of comprehensive documentation

2.2.4.3 V-model

In 2023, Hamilton has posted the information about V model on the website. According to his information, V-model also known as Validation or Verification Model. In addition, it is the extension of the waterfall model. In this model, the testing phase is parallel to each development phase. In short, it highlights the relationship between each phase of development and its associated testing phase.

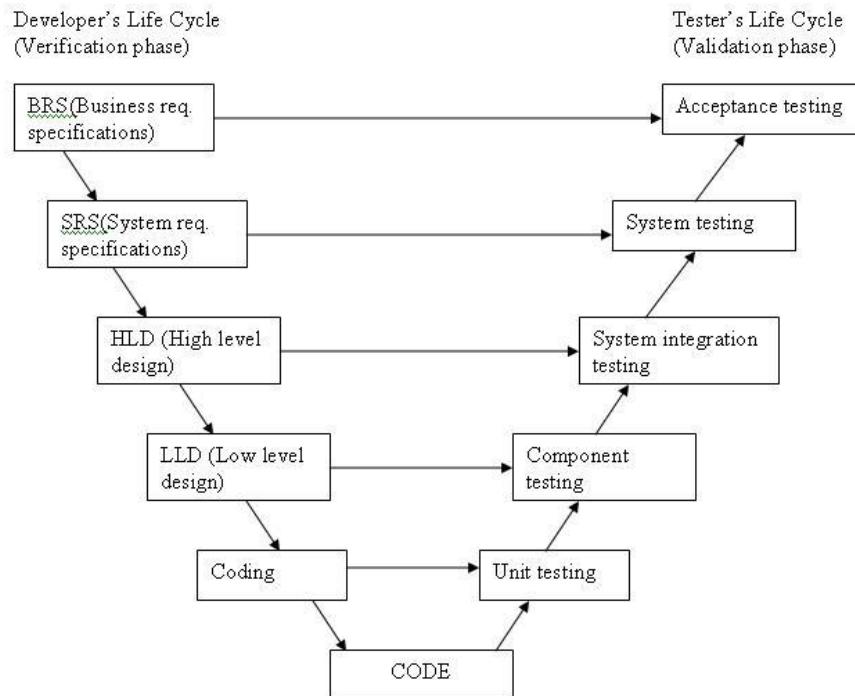


Figure 2. 3 V-Model (TRY QA, n.d.)

- Advantages:
 - Clear correspondence between development phase and testing phase
 - Early test planning
 - Through testing
 - Traceability
- Disadvantages:
 - Not agile
 - Limited user involvement
 - Inflexible to changes in requirements

2.2.4.4 Spiral model

Martin stated that the Spiral model is risk-driven and combined from waterfall model and iterative model (2023). In more specific, each phase is begun with a design goal and ended with the client reviewing the progress. The process is developed and released incrementally and each of the deliverable is known as a phase or a loop of the spiral. The phases in this model including planning, risk analysis, engineering, and evaluation.

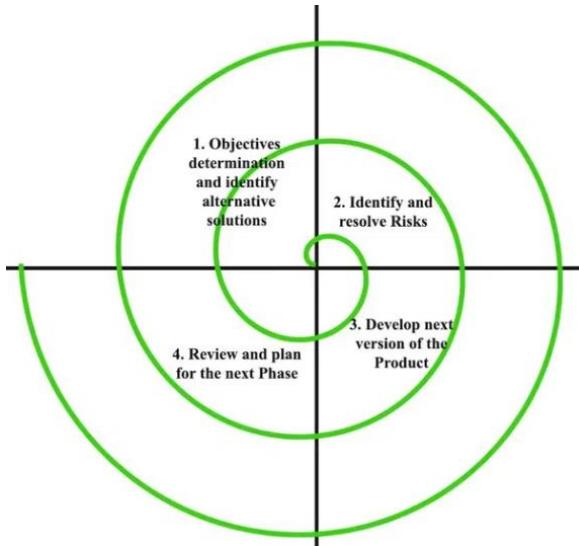


Figure 2. 4 Spiral Model (Martin, 2023)

- Advantages:
 - Risk management
 - Flexibility
 - Customer feedback
 - Earlier defects identification
- Disadvantages:
 - More complex
 - Resource intensive
 - Not suitable for small projects
 - Extremely place importance on documentation

2.2.4.5 Prototype model

Prototype is a type of software development model that creates partial or representative prototypes to gather feedback and refine requirements before the full system is developed (Lewis, 2023). In the iterative development process of the new system, detailed requirements are gathered through user interviews, followed by the creation of a preliminary design. The initial prototype is then constructed, and users thoroughly evaluate it, providing feedback for modification. This cycle is repeated until the prototype aligns with the final product desired. The final system is then built based on the refined prototype, undergoes comprehensive evaluation and testing, and is subject to routine maintenance for continued reliability.

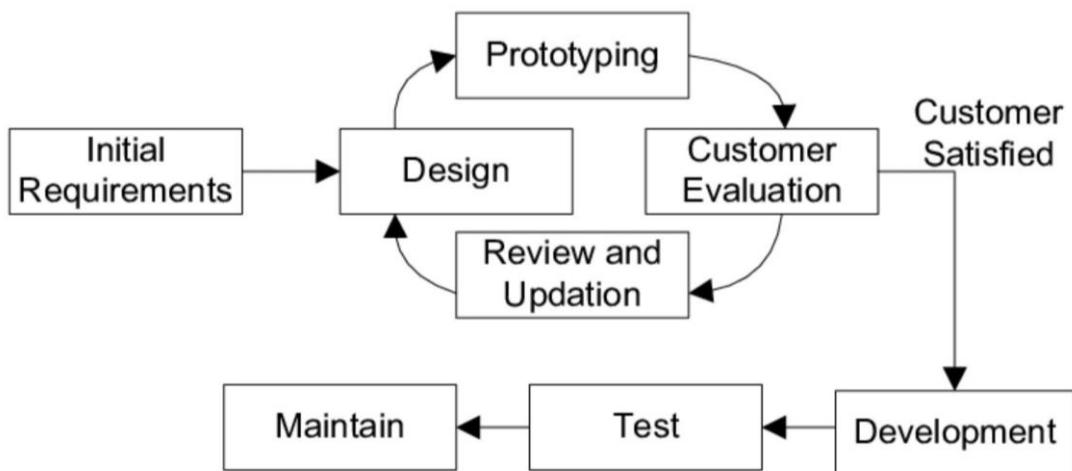


Figure 2. 5 Prototype Model (Infinity and Board Infinity, 2023)

- Advantages:
 - Better customer and user involvement
 - Better communication
 - Early issues identification
 - Flexible and adaptable to changes in requirements
 - Disadvantages:
 - Incomplete system
 - Higher potential for scope creep
 - Limited scalability
 - Highly dependent on user involvement

2.2.5 Research Instrument

2.2.5.1 Questionnaire

Questionnaire is one of the most common used instruments to gather the information. It consists of various types of questions to collect and gather the data that is related to the research topic (Bhat, 2023b). A research questionnaire is often composed of close-ended and open-ended questions.

- Advantages:
 - Cost-effective
 - Wide geographic reach
 - Quantifiable data
 - Structured data collection
- Disadvantages:
 - Low response rates
 - Limited flexibility
 - Sampling bias

2.2.5.2 Observation

Observation is a technique to gather data by watching people, events, and the environment that the researchers want to research about. Normally, the researchers are required to immerse themselves in the environments that they want to research. Observations could be divided into 3 types: participant observation, direct observation, and indirect observation (Duke University Libraries, n.d.).

- Advantages:
 - Unobtrusive data collection
 - Naturalistic settings
 - Behavioral Insights
 - Real-time data collection
- Disadvantages:
 - Subject reactivity
 - Observer bias
 - Time consuming

2.2.5.3 Interview

George stated that the interview is a helpful technique to collect data by direct asking questions to the interviewees (2023b). There are three types of interviews can be used: unstructured, semi-structured, and structured interviews. Normally, it means the degree of formal in interview session. Researchers can choose the suitable interview approach based on their case and requirement.

- Advantages:
 - In-depth information
 - Flexibility
 - Personal connection
 - Tailored to different groups of interviewees or stakeholders
 - Real-time interaction
- Disadvantages:
 - Time consuming
 - Social desirability bias
 - Resource intensive
 - Limited sample size

2.2.5.4 Focus Group

This instrument allows the researchers to gather the information and data by gathering a small group of participants to answer the pre-defined questions or converse naturally and physically (George, 2023c). This is known as a type of qualitative research.

- Advantages:
 - Share diverse perspectives among group members
 - Rich data collection
 - Suitable to explore complex topics
 - Immediate feedback
- Disadvantages:
 - Group influence
 - Time consuming
 - Not suitable for sensitive topics

2.2.5.5 Experiment

Bath mentioned that experiment is a research instrument that enables the researcher to control and manipulate the independent variables and estimate the effects on dependent variables (2023a). Therefore, it serves as an important technique to understand the cause-and-effect relationships. In this instrument, a hypothesis is important to be set and tested to verify the relationships and assumptions. It means that the outcomes of dependent variables are eventually measured to prove the validity of the hypothesis.

- Advantages:
 - Controllable conditions
 - Quantitative data
- Disadvantages:
 - Complex process
 - Higher cost
 - Potential for bias
 - Skill dependency

2.2.6 Research Design

2.2.6.1 Qualitative Research

This research method intentionally involves a smaller number of participants, emphasizing in-depth interactions that, albeit resource-intensive in terms of time and costs, offer a comprehensive exploration of attitudes, behaviors, and experiences within a specific area (Tenny, 2022). In this approach, the researcher may be personally known to the participants, and participant characteristics remain transparent. The goal is to gain profound insights, employing techniques like interviews, focus group discussions, and participant observations. Data collection encompasses various forms such as words, images, and objects, with the aim of formulating hypotheses and theories based on the acquired understanding.

- Advantages:
 - In-depth understanding of phenomenon
 - Flexible nature
 - Suitable for exploration of complex phenomena
 - Open-ended exploration
- Disadvantages:
 - Subjectivity and bias
 - Time consuming
 - Resource intensive
 - Difficulty in combining quantitative data

2.2.6.2 Quantitative Research

This research method focuses on the collection of quantitative data using various techniques (Bhandari, 2023). To enhance data reliability, it often involves a large-scale, random selection of participants, minimizing direct interaction with objects compared to qualitative methods. Typically, participants are unfamiliar with the researcher, and their characteristics remain undisclosed. Common techniques include questionnaires or structured interviews. The data obtained are numerical and statistical, facilitating the identification of correlations among variables. This type of research allows for hypothesis testing, playing a crucial role in predicting outcomes and guiding the overall project direction.

- Advantages:
 - Large sample size
 - Statistical analysis
 - Well suited for hypothesis testing
- Disadvantages:
 - Limited exploration

- Time consuming in data analysis

2.2.6.3 Mixed-methods research

George defined the mixed methods research as a combination of quantitative research and qualitative research (2023A). Since the nature of combination from both research methods, it is more comprehensive.

- Advantages:
 - Comprehensive understanding
 - Enhance validity of the research
 - In-depth exploration
 - Iterative process
- Disadvantages:
 - Complexity
 - Resource intension
 - Increase workload
 - Highly dependent on researcher competence

2.2.7 Existing System

2.2.7.1 Magaya Supply Chain System

Magaya offers a versatile Supply Chain Management (SCM) solution suitable for businesses of various sizes, including microbusinesses and SMEs. Despite not being furniture-specific, its automation streamlines inventory management, order processing, and warehouse optimization. Magaya ensures accuracy in order fulfillment and integrates seamlessly with accounting and e-commerce systems. Particularly beneficial for smaller businesses, its automation reduces manual work, providing a cost-effective solution for growth (TrustRadius, 2023). The system is reasonably priced and scalable to accommodate evolving business needs.

2.2.7.2 FreightPOP

FreightPOP specializes in comprehensive freight management and logistics, catering to businesses heavily reliant on shipping. Key features include Freight Rate Management for cost-effective carrier selection, real-time tracking for enhanced visibility, and support for various transportation modes. The platform optimizes costs through efficient route and carrier choices, benefiting microbusinesses and SMEs by enhancing shipping efficiency and contributing to cost savings (TrustRadius, 2023b). Reliable shipping processes also positively impact customer satisfaction.

2.2.7.3 Precoro

Precoro specializes in streamlined procurement and vendor management, offering features like Purchase Requisitions for efficient approval workflows. The platform supports vendor tracking, performance evaluation, and budget control, allowing spending limits and continuous expense monitoring (Boehm, 2023). Integration with accounting software and ERPs enhances functionality, making it advantageous for microbusinesses and SMEs. Precoro ensures timely approvals, reducing delays, and contributes to cost control through effective budget and vendor management, providing transparency into procurement activities

2.2.7.4 IBM Blockchain Supply Chain Solution

IBM also provides the SCMS which is blockchain-based for business to use. It serves as a BaaS (Blockchain-as-a-service) solution thereby providing potential customers with the flexibility in terms of adapting to the specific needs. The provided components involve Supply Chain Intelligence Suite and Order Management System. They support the tasks like digital verification for procurement, food system and so on (IBM,2023). This system focuses on dealing with the data visibility, process optimization, demand management thus improving the whole supply chain from root. The blockchain of IBM SCMS is built based on Hyperledger Fabric 2.4. It fully utilizes the modular architecture of Hyperledger Fabric. It allows a more robust, faster development process but also significant support from the social since it is an open source blockchain platform. The modularity also provides a benefit

which is flexibility in choosing consensus protocols to be suit with specific business requirements (IBM, 2022).

2.2.7.5 Summarized Table of Existing System

<i>Summary of Existing System</i>	
Name	Features
Magaya Supply Chain System	<ul style="list-style-type: none"> • Offerin a versatile SCM solution suitable for businesses of various sizes, including microbusinesses and SMEs. • Not furniture-specific, but its automation streamlines inventory management, order processing, and warehouse optimization. • Ensures accuracy in order fulfillment and integrates seamlessly with accounting and e-commerce systems. • Particularly beneficial for smaller businesses, reducing manual work and providing a cost-effective solution for growth. • Reasonably priced and scalable to accommodate evolving business needs.
FreightPoP	<ul style="list-style-type: none"> • Specializing in comprehensive freight management and logistics. • Key features include Freight Rate Management for cost-effective carrier selection. • Real-time tracking enhances visibility, supporting various transportation modes. • The platform optimizes costs through efficient route and carrier choices. • Benefits microbusinesses and SMEs by enhancing shipping efficiency and contributing to cost savings. • Reliable shipping processes positively impact customer satisfaction.
Precoro	<ul style="list-style-type: none"> • Specializing in streamlined procurement and vendor management. • Features include Purchase Requisitions for efficient approval workflows. • Supports vendor tracking, performance evaluation, and budget control. • Allows spending limits and continuous expense monitoring. • Integration with accounting software and ERPs enhances functionality.

	<ul style="list-style-type: none"> • Advantageous for microbusinesses and SMEs. • Ensures timely approvals, reducing delays. • Contributes to cost control through effective budget and vendor management. • Provides transparency into procurement activities.
IBM Blockchain Supply Chain Solution	<ul style="list-style-type: none"> • Providing SCMS, a blockchain-based solution for businesses, serving as a BaaS (Blockchain-as-a-service) offering. • Offers flexibility to adapt to specific business needs. • Components include Supply Chain Intelligence Suite and Order Management System. • Supports tasks like digital verification for procurement and food systems. • Focuses on data visibility, process optimization, and demand management. • Improves the entire supply chain from the root. • Built on Hyperledger Fabric 2.4, utilizing its modular architecture. • Enables robust, faster development processes and significant community support as an open-source blockchain platform. • Offers flexibility in choosing consensus protocols to suit specific business requirements.

Table 2. 2 Summarized Table of Existing System

2.2.8 Review of Technical Domain

2.2.8.1 Blockchain

The integration of blockchain technology stands as a foundational element in the development of the envisioned Supply Chain Management System (SCMS). Central to this initiative is the decision to employ blockchain as a notary within the SCMS framework, capitalizing on its inherent attributes of security, transparency, and tamper resistance. By securely anchoring crucial information on the blockchain, particularly tailored to the operational dynamics of Small and Medium-sized Enterprises (SMEs) engaged in the customization furniture industry and operating under Make-to-Order (MTO) processes, the project aims to fortify data integrity and immutability.

2.2.8.2 Smart Contract

In considering the role of smart contracts within this context, the inherent variability in orders stemming from the MTO nature of the SME-scale customization furniture industry necessitates a flexible approach. From this point, the decision to exclude smart contracts aligns with the project's strategic focus on accommodating the dynamic and bespoke production requirements characteristic of the industry. However, the existence of smart contract can facilitate the process of recording information to the blockchain. It represents reasonable approach how this system will record data to blockchain.

2.2.8.3 Blockchain Platforms

The exploration of suitable blockchain platforms involved a comprehensive evaluation of options such as Ethereum, IBM Blockchain, and Hyperledger Fabric. The selection criteria were meticulously crafted, taking into account factors such as community support, features, scalability, and alignment with the project's overarching goals. In this deliberation, the Ethereum blockchain emerged as the preferred platform, distinguished by its robust smart contract capabilities and widespread adoption. This choice underscores its suitability for the decentralized applications integral to the envisioned SCMS.

2.2.8.4 Programming Languages, Tools, and Frameworks

The selection of programming languages for both the web-based Supply Chain Management System (SCMS) and blockchain system underscores a commitment to creating an accessible, user-friendly, and adaptable platform. JavaScript, renowned for its versatility and widespread support, plays a pivotal role in enhancing the interactive and dynamic facets of the user interface. The combination of HTML and CSS contributes to the system's structural integrity and styling.

Express.js and EJS are integral to the SCMS development. Express.js, a minimal and flexible Node.js web application framework, simplifies server-side development with robust features for handling HTTP requests and routing. EJS, a templating engine, allows for the embedding of JavaScript within HTML templates, facilitating the creation of dynamic web pages

In terms of tools and frameworks, Web3.js emerges as a crucial component in the development ecosystem. As a JavaScript library, Web3.js plays a key role in facilitating interactions with the Ethereum blockchain. It enables the seamless creation and signing of transactions, thereby fostering effective communication between the web based SCMS and the decentralized network. This choice aligns harmoniously with the project's overarching objective of achieving a seamless integration of web development with blockchain interactions.

To further bolster the development stack, the inclusion of Truffle, Ganache, and Solidity is indispensable for the blockchain aspect of the SCMS. Solidity, as the primary programming language for writing smart contracts on the Ethereum blockchain, offers syntax and semantics specifically designed for creating complex contractual agreements and logic in a secure and efficient manner. Truffle provides a development environment, testing framework, and asset pipeline for blockchains using the Ethereum Virtual Machine (EVM), while Ganache offers a personal blockchain for rapid development and testing

Additionally, the integration of Node.js and Bootstrap contributes to the holistic development environment. Node.js, serving as a server-side runtime for executing JavaScript, enhances the overall efficiency of web applications. Bootstrap, a prominent front-end framework, streamlines design processes and ensures responsiveness. Together, these tools and frameworks constitute a robust foundation for the development and functionality of the SCMS.

2.2.8.5 Software Development Methodology

The adoption of the Agile software development methodology underscores the project's commitment to flexibility, collaboration, and adaptability. Agile principles, such as iterative development and continuous feedback, align with the dynamic nature of supply chain management in the SME-scale customization furniture industry.

2.2.8.6 Future Improvements

The project presents a visionary roadmap, closely aligned with the rapidly evolving landscape of blockchain technology and the dynamic needs of Small and Medium-sized Enterprises (SMEs) in the customization furniture industry. It addresses the prevalent challenges, such as limited digitalization and the industry's gradual transition from Industry 2.0 practices, aiming to catapult SMEs into the forefront of Industry 4.0. As blockchain platforms advance, the planned updates and integrations are meticulously designed to tackle essential aspects like scalability, interoperability, and enhancing smart contract functionalities.

Incorporating smart contracts from the beginning is a strategic initiative, primarily focused on leveraging their capability to securely and immutably record data on the blockchain. This decision is instrumental for laying the foundation for future enhancements, which will be strategically directed towards harnessing blockchain-based automation and efficiency. It signifies a pivotal move towards

facilitating SMEs in adopting a more advanced and digitally transformative operational framework. The utilization of blockchain technology, especially through the application of smart contracts for data recording, is chosen for its unparalleled ability to serve as a secure and immutable ledger. This is vital for instilling transparency and trust in supply chain transactions, directly addressing some of the industry's most significant challenges.

As the project progresses, the specific advantages of employing blockchain technology and smart contracts for data recording in addressing the unique challenges faced by the industry will become increasingly apparent. This strategic application is anticipated to streamline operations significantly, thereby enhancing the resilience and competitiveness of SMEs in the customization furniture sector. By adopting this advanced technological approach, the project aims to lead a transformative shift towards digital excellence and operational efficiency, marking a new chapter of innovation and growth for SMEs.

Chapter 3 Preliminary Study

3.1 Overview

In the previous chapter, the research makes significant contribution for the rest of parts of this research. It reveals the business process of object, which is small-scale customization furniture industry. Besides that, the technologies and methods that can be used for the development of SCMS have been researched as well.

The quantitative research method will be applied to strengthen the findings that have been obtained from the literature review. In this case, it will directly affect the results of system attributes, scale and so on. Quantitative research is the process of implementing strategies to collect the data that relates to research and analyze the collected data (Wilson, 2019). The literature review also contributes to the primary data collection in terms of developing suitable questions to gain the information.

To ensure the collected data will highly relate to this research, the questionnaire will be used to collect the data on such industry's workers on the SCMS. Their opinions will be the valuable information for this research. The targeted group of people will be the customization furniture industry' employers, employees, interns, suppliers, and so on.

3.2 Project Research Design

3.2.1 Design Science Research Methodology (DSRM) Table

<i>Design Science Research Methodology</i>	
<i>Phase</i>	<i>Detail of activities</i>
Phase 1	<p>Identification of problem and motivation</p> <ul style="list-style-type: none">• Based on the literature review, we identified the SME scale customization furniture industry lack of the system to manage their supply chain properly due to the cost, knowledge, and so on. This is a gap for us to fill it up.• Besides that, the current academic articles that discuss this industry is rare.• A preliminary study is conducted to judge whether the proposed system can help the involvers in such industry to improve their business operations.• The results of the initial study on the problem identified a suitable SCMS can be helpful in doing the preliminary digital transformation in this industry.
Phase 2	<p>Define Objectives of the Solution</p> <ul style="list-style-type: none">• The findings from the literature review have provided insights into the supply chain management problems within this industry that require attention. Additionally, they have shed light on potential technologies and

	<p>methods that can be employed in the development of an effective supply chain management system (SCMS).</p> <ul style="list-style-type: none"> • The development of the conceptual framework is done to visualize how the system works, which serves as the reference for the design phase.
Phase 3	<p>Design and Development</p> <ul style="list-style-type: none"> • The characteristics and element of the system will be evaluated by translating the flowchart into a hierarchical task analysis model. • The opinions from the system's potential user are undertaken to ensure the system is usable and satisfies the needs of users. • To facilitate the development process, experimental methods will be incorporated to gain a deeper understanding of the necessary features that the system should encompass.
Phase 4	<p>Demonstration</p> <ul style="list-style-type: none"> • An evaluation criterion is developed to validate the artifact. • To obtain user feedback on the artifact.
Phase 5	<p>Evaluation</p> <ul style="list-style-type: none"> • The user testing is being used for getting the feedback from the potential users. The participants will be the people who are involved in this industry. • Performance measures include learning effectiveness and participant's satisfaction.
Phase 6	<p>Communication</p> <ul style="list-style-type: none"> • All results and findings from prior literature review, preliminary study, data analysis, artifact design, evaluation reports, model validation report were combined and written into a full comprehensive report.

Table 3. 1 Design Science Research Methodology (DSRM)

3.2.2 Project Design Table

NO	Research Question	Research Objectives	Method	Expected Outcome
1	What are the existing technologies and methods used in supply chain management, particularly for SMEs and microbusinesses?	To study existing technologies and methods used in supply chain management system.	<ul style="list-style-type: none"> • Literature review • Study existing system 	List of existing technologies used in SCMS by different researchers.
2	How does the SCMS manage the flow and movement of goods to enhance the supply chain efficiency?	To design and develop a Supply Chain Management System on the customization furniture industry.	<ul style="list-style-type: none"> • Questionnaire 	Develop a prototype of supply chain management system (SCMS) for customization furniture industry to use.
3	Will the adoption of a tailored SCMS enhance the operational efficiency of customization furniture industry?	To validate the efficiency and effectiveness of the Supply Chain Management System on the customization furniture industry.	<ul style="list-style-type: none"> • Experimental • Conduct User Acceptance Testing (UAT) 	To validate the usability and reliability of the system.

Table 3. 2 Project Design

3.2.3 Evaluation Table

<i>Supply Chain Management System for SME scale Customization Furniture Industry</i>	
Participant	People who involve in this industry for example, employers, employees, secretary, interns and so on.
Task	Log in, sign up, choose supplier, display product information, place order, generate invoice
Techniques	Study existing technologies, questionnaire, experiment
Measurement	Ease of use, usability, and reliability of the system
Outline	<ul style="list-style-type: none"> • Explain the goal and objective of the evaluation • Demonstrate the system to the participants • Provide questionnaire to the participants to collect their point of view for system improvement • Transcribe collected data into a summarized data • Analyze the questionnaire data

Table 3. 3 System Evaluation Table

3.3 Operational Framework

Item	Phases	Activity Description	Output	Expected Contribution
Idea	Phase 1: Problem Identification and Motivation	<ul style="list-style-type: none"> Conduct literature review (Existing technologies, business processes, etc) Conduct preliminary studies and research 	Conceptual Model	<ul style="list-style-type: none"> Knowledge on building a SCMS for customization furniture industry.
Conceptual Model	Phase 2: Suggestion and Solutions	<ul style="list-style-type: none"> Literature review Survey 	<ul style="list-style-type: none"> Refined Conceptual Model 	<ul style="list-style-type: none"> Artifact – Prototype of SCMS Technique to detect and categorize of user activity Mechanisms to provide punctual prompt of user activity
Artefact (Prototype)	Phase 3: Artefact Design and Development		<ul style="list-style-type: none"> Artifact Design 	
Framework	Phase 4: Artefact Evaluation	<ul style="list-style-type: none"> Develop reliable measures assessment Develop test cases to measure the performance Users use the artefact and evaluate the system's performance 	<ul style="list-style-type: none"> Identified the Performance Measures 	
	Phase 5: Prototype Evaluation	<ul style="list-style-type: none"> To conduct experiment with working prototype 	<ul style="list-style-type: none"> Validate Model 	<ul style="list-style-type: none"> Final Model of SCMS for customization

				furniture industry
	Phase 6: Documentation	<ul style="list-style-type: none"> • To combine all results from literature review, preliminary study, artefact design, artefact evaluation report, framework validation report into a final report 	<ul style="list-style-type: none"> • Final Thesis Report 	

Table 3. 4 Operational Framework

3.4 Project Research Instruments

In this research, there are two instruments will be utilized which are questionnaire and experiment. The questionnaire was conducted among the involvers in SME-scale customization furniture industry in order to get their opinions of implementing the SCMS. From the perspective of experiment, it was used for the validation of usability and reliability of the system.

There were several forms of questions had been included in the questionnaire. The following table demonstrates various forms of used questions:

<i>Different Formats of Questions in Questionnaire</i>		
No	Type	Explanation
1	Close-ended Questions	Close-ended questions are designed to elicit specific, limited responses. These questions typically have a finite set of response options, such as "yes" or "no," or multiple-choice options. They are often used to gather structured and easily quantifiable data.
2	Open-ended Questions	Open-ended questions encourage respondents to provide detailed, free-form answers or opinions. These questions do not limit respondents to predefined choices and allow for more in-depth responses. Open-ended questions are often used when to gather qualitative data and explore the research in depth.
3	Demographic Questions	Demographic questions gather information about respondents' personal characteristics, such as age, gender, income, education, and location. These questions help categorize and segment survey data to analyze how different groups of people respond to various question.
4	Single-answer Multi Choice Questions	Single-answer multi-choice questions offer respondents multiple options to choose from, but they can select only one answer. These questions are useful for capturing a single, specific response.
5	Rating Scale Questions	Rating scale questions ask respondents to rate a statement or item on a scale, typically from low to high. This allows to quantify opinions or perceptions. The scale may vary in length, but common ones include 1-5, 1-7, or 1-10.
6	Likert Scale Questions	Likert scale questions are a specific type of rating scale that uses a range of ordered response options, such as "strongly agree," "agree," "neutral," "disagree," and "strongly disagree." Respondents choose the option that best reflects their opinion or agreement level.

Table 3. 5 Different Formats of Questions in Questionnaire

3.5 Data Collection

3.5.1 Population and Sample

The population group for this research are the involvers in SME-scale customization furniture industry. The target population can be accessed through giving out questionnaires to the users by using the email, social media, such as Facebook, Instagram, and so on. The participants could be whoever come from different states across Malaysia.

Participants	Employers, employees, secretary, interns in SME-scale customization furniture industry
Technique	Questionnaire
Representative tasks	Analyze the data with the use of a database and computer systems
Measurements	Functionality of system, reliability of system, and ease of use
Outline plan	Questionnaire preparation, questionnaire distribution, responses collection and data analysis

Table 3. 6 Population and Sample

3.6 Chosen Software Development Lifecycle

SDLC refers to Software Development Lifecycle, which is a process to creates the software in a systematic manner in order to improve the quality and reduce the cost of final product (Altavater, 2023). The processes involved in SDLC may include requirement analysis, planning, software design, software development, testing, and deployment.

Meanwhile, the software development methodology that had been identified in previous chapter represents the alternative of different types of SDLC, which is also known as SDLC model or methodology (Choudhary, 2023). They are Waterfall model, Agile model, V-model, Spiral model, and prototype model. Based on the research before, each of them has different pros and cons.

The chosen SDLC methodology for this project were agile methodology. Agile methodology provided various advantages for this project such as:

- Adaptability to changes:

It allows for flexibility and adaptability throughout the development process. Especially the literature resources are insufficient in this area, the ability to adjust to changing requirements or new insights is crucial.

- Incremental development:

Agile emphasizes incremental and iterative development. This is beneficial when dealing with a lack of comprehensive literature resources.

- Customer-centric approach:

Agile focuses on delivering value to the customer. In this project addressing SME-scale customization furniture industry needs, this approach ensures that the system's functionalities, such as sign up, log in, supplier selection, order placement, and invoice generation, directly address the practical requirements of the users.

3.6.1 Proposed Agile Model

The Agile model consists of six distinct phases. The following figure are the graphical representations of six phases and the corresponding tasks in this model for this project:

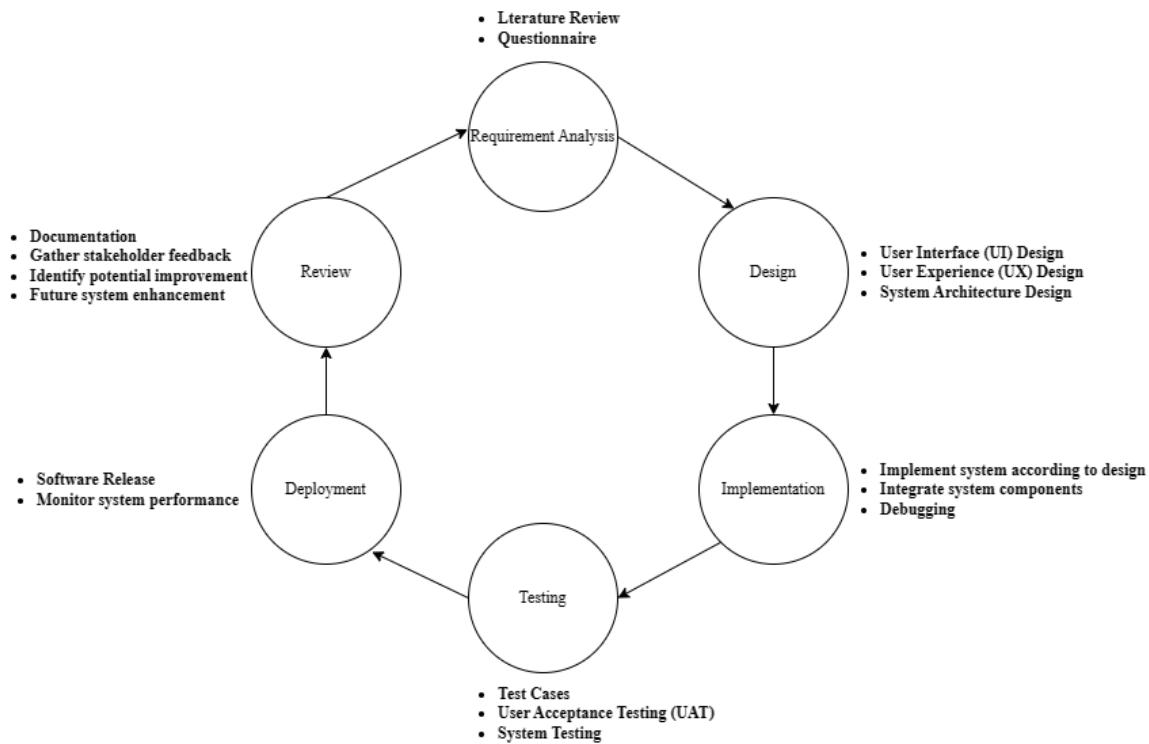


Figure 3. 1 Proposed Agile Model

Chapter 4 Analysis

4.1 Introduction

A survey is conducted to delve into the perception of potential users from various perspective, such as the willingness, acceptance, and awareness of the proposed system. These potential users involve employers, production teams, sales and marketing teams, secretary, and interns who involve in this industry. Additionally, the aimed to validate the validity of issues related to the ordering process in such industry, such as the frequency of problems and ordering occurrences. The validation process ensured the usefulness of the proposed system. The questionnaire was distributed through Facebook group, WhatsApp, and email. To prioritize respondent confidentiality, participants have the option to choose whether to disclose their company name when answering the questionnaires. The anonymous nature of survey allows the respondents to provide the accurate information that is exactly derived from their mind without other consideration. This advantage consequently contributes to the reliability and useability of the questionnaire.

4.2 Questionnaire Details

4.2.1 General Information – Demographic

Section A – Demographic Details	
Question	Purpose
Company Name (Optional)	To explore which company has involved in this questionnaire.
Role in the furniture industry	To explore what are the role of the company in the industry.
Role in the company	To provide insights into the role of respondent in their company.
Year of the company in the furniture industry	To explore the experience of respondent's company involves in the company.
Size of the company	To provide insights into the size of respondent's company.

Table 4. 1 Demographic Details

4.2.2 Experience on Ordering Process from Supplier and Customer.

Section B – Experience on Ordering Process from Supplier and Customer	
Question	Purpose
How frequently the order is placed to your supplier?	To explore the frequency of an order is place.
How do you currently manage your ordering process? (Multiple choice)	To explore the way of company in handling ordering process.
What are the main challenges you face in the current ordering or order fulfilment process? (Multiple choice)	To provide insights into the challenges that are faced by company.
How often did the mistake occur in the realm of ordering? (If you are supplier, please answer from your perspective and your customer)	To explore the frequency of an order having problem
How effective do you find your current supply chain management system in meeting your company's needs?	To explore the satisfaction of respondent towards the current ordering process in company
How serious were the production impacted by the ordering problem?	To identify how serious is the ordering problem or mistake in impacting the productivity

Table 4. 2 Experience on Ordering Process from Supplier and Customer

4.2.3 Question on Blockchain Supply Chain Management System

Section C – Question on Blockchain Supply Chain Management System	
Question	Purpose
What factors influence your decision to adopt new technologies in your supply chain management? (Multiple choice)	To identify the reasons that can enforce the company to apply the supply chain management system
Do you think a material ordering system can improve your company productivity by improving the ordering process?	To determine whether the system can really be helpful from the perspective of respondent

Are you familiar with blockchain technology and its applications in the furniture industry?	To assess whether the respondent was aware of blockchain technology and the application in this industry.
Do you think the blockchain technology can increase the confidence of each transaction with the supplier and customer?	To assess perceptions of blockchain ability to enhance trust and reliability between supplier and customer (factory)
How willing would you be to adopt this supply chain management system to your business?	To provide insights into the willingness of the respondent in using the new system

Table 4. 3 Question on Blockchain Supply Chain Management System

4.3 Results of the Overall Findings

Company Name (Optional)

0 responses

No responses yet for this question.

Figure 4. 1 Company Name

This is the first question were asked for the company name of respondent. Surprisingly, out of 105 respondents, none were willing to reveal their company name.

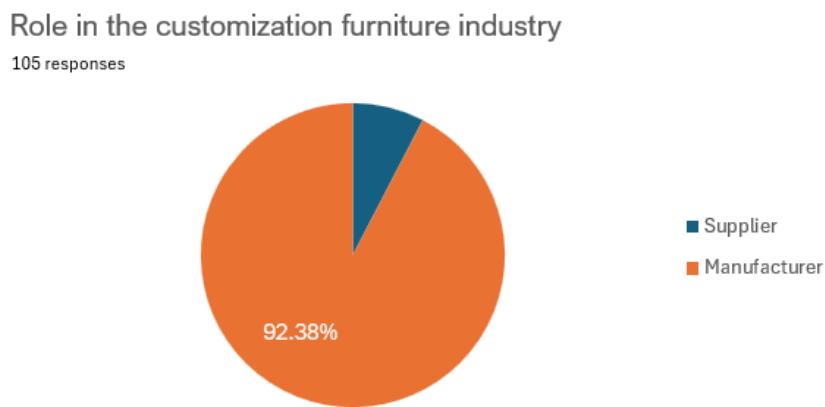


Figure 4. 2 Role in Industry

This pie chart reveals the distribution of roles within the customization furniture industry based on the survey with 105 responses. The distribution of survey respondents between these two roles is notably uneven. There are 92.3% of respondents (97 individuals) the manufacturer, and the rest of respondents are the representative from the supplier.

Role in the company

105 responses

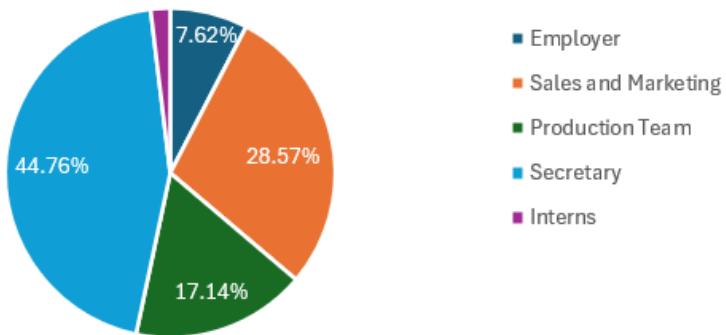


Figure 4. 3 Role in Company

This question seeks to ascertain the various roles of respondents within their respective companies. The majority of respondents, constituting 44.76%, hold the position of secretary. Following closely, 28.57% of respondents are affiliated with the sales and marketing sectors, while approximately 17% serve in production roles within their companies. Notably, a mere 8% of respondents identify as employers. Interestingly, 2% of respondents are interns in their respective companies.

Year of the company in the customization furniture industry

105 responses

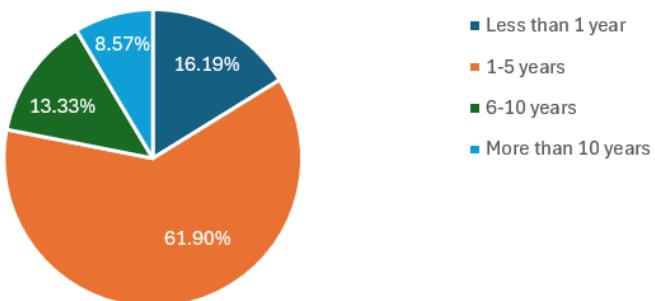


Figure 4. 4 Year in Industry

The pie chart illustrates the distribution of companies by their years of involving in customization furniture industry. Out of 61.90% of companies have been operating in the industry within 5 years but more than 1 year, this is also the largest segment in the pie chart. The second-largest group, representing 16.19% of the companies, are those that have been in the industry less than 1 year. There are 13.33% of the companies fall into the category of having been in business for 6-10 years. Lastly, around 9% of newest entrants have involved in this survey since their business is not more than 1 year.

Size of the company

105 responses

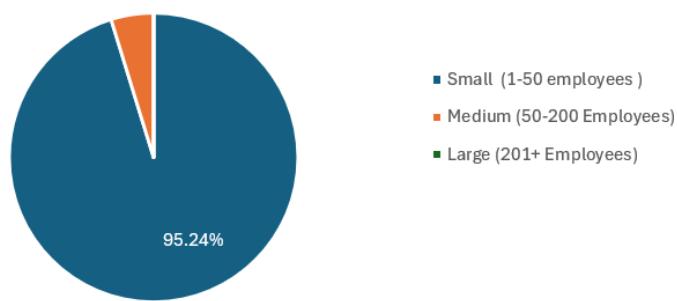


Figure 4. 5 Company Size

This chart depicts the distribution of the sizes of companies within the customization furniture industry, based on 105 responses. A vast majority, 95.24% of the companies surveyed are classified as small, with a workforce, ranging between 1 to 50 employees. The rest of 4.76% responses, whereby represent the companies which have more than 50 but less than 200 employees. Obviously, there is no company has more than 200 employees involved in this survey.

How frequently the order is placed to your supplier or from your customer?

105 responses

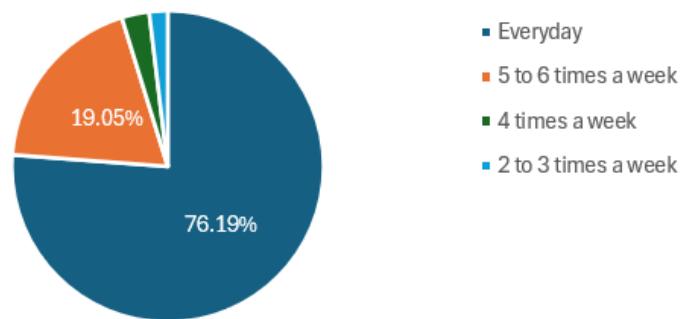


Figure 4. 6 Frequency of Placing Order

This question was used to understand the needs of respondents' company in terms of placing order within a week. Based on 105 responses, most of the responses, which is 76.19%, indicated that the order will be placed in every day. A smaller portion of responses, 19.05%, mentioned that orders are placed mostly every day, equating to 5-6 times a week. Both categories, which are 4 times a week and 2 to 3 times a week, only took the small portion in this survey. They are 3% and 2% respectively. This indicates that the MTO is exactly applied in this industry.

How do you currently manage your ordering process?

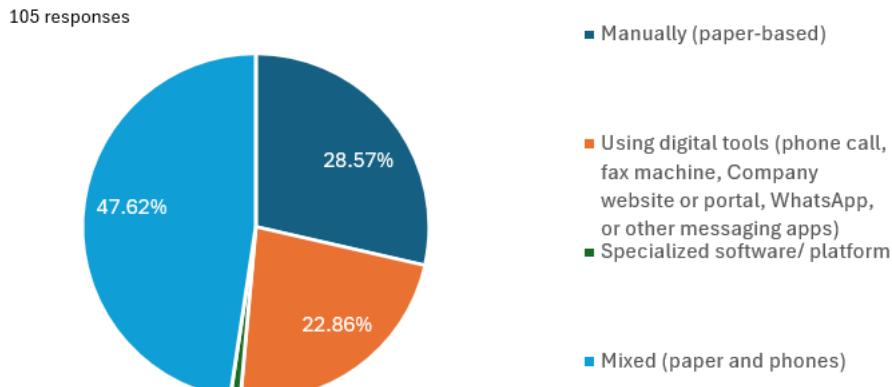


Figure 4. 7 Way to Manage Ordering Process

The pie chart illustrates that 47.62% of companies in the customization furniture industry prefer a mixed approach, utilizing both paper-based systems and digital tools, as the most prevalent method for managing the ordering process. A significant number of companies, 28.57%, still manage their orders manually, suggesting a potential area for technological improvement. The smallest group, at 22.86%, has adopted digital tools, indicating a trend towards modernization, although it remains less prevalent. This insight into the current state of order management practices can inform strategies for implementing blockchain technology within the industry.

What are the main challenges you face in the current ordering or order fulfilment process?

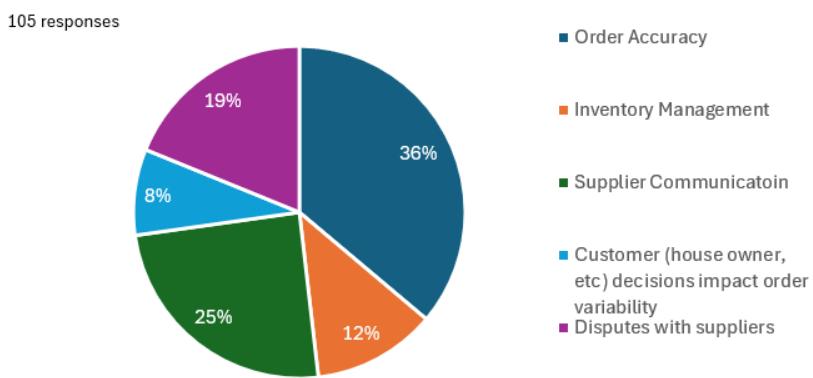


Figure 4. 8 Challenges in Managing Ordering Process

This is a multi-choices question for the respondents to choose. The results show that the biggest challenge, at 36%, is order accuracy, followed by supplier communication, which is 25%. Additionally, 19% of respondents have experienced the disputes with supplier. 12% of respondents also experienced the problem in terms of inventory management. Lastly, the impact of customer decision on order variability is the least, yet it still has 8% of respondents vote for this.

How often did the mistake occur in the realm of ordering? (If you are supplier, please answer from your perspective and your customer)

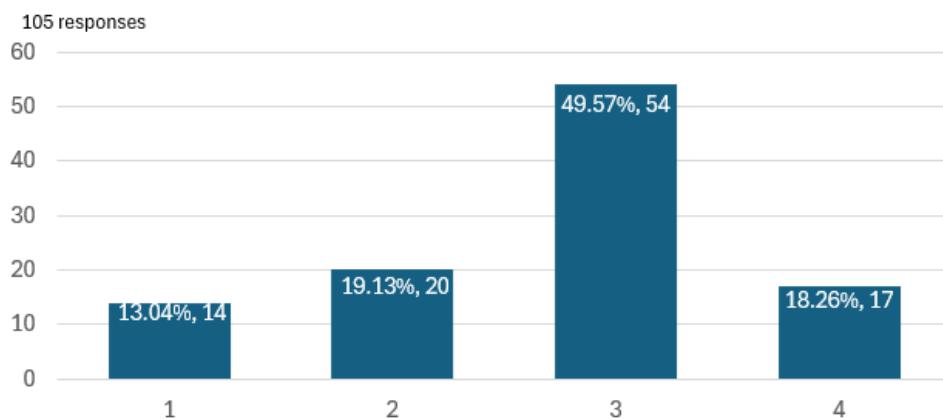


Figure 4. 9 Frequency of Mistake Occurring

The survey results indicate the likelihood of errors in the ordering process. 13.04% of respondents, which is 14 individuals, believe that the mistake is not happening frequently. Around 19% of respondents experience a slightly higher frequency of mistakes. However, it is still categorized in the “unlikely category”. A significant majority of respondents (49.57%) frequently encountered mistakes in ordering. 17 respondents, which is around 18.26%, encountered the mistake in a high frequency. Overall, there are 71 respondents frequently face a mistake, this is significantly more than the respondents who did not face a mistake in the realm of raw material ordering.

How effective do you find your current supply chain management system in meeting your company's needs?

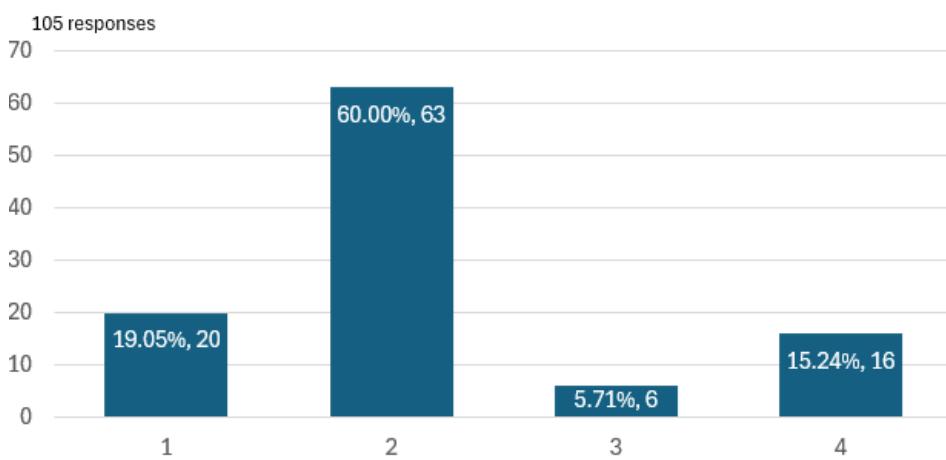


Figure 4. 10 Effectiveness of SCMS

The bar chart reflects the perceived effectiveness of current Supply Chain Management among 105 respondents. 63% of respondents rated their system as having limited effectiveness in meeting their needs. About 19% of respondents expressed dissatisfaction with their system. Out of 105 responses, there were only 22 respondents rate for satisfaction with their system.

How serious were the production impacted by the ordering problem?

105 responses

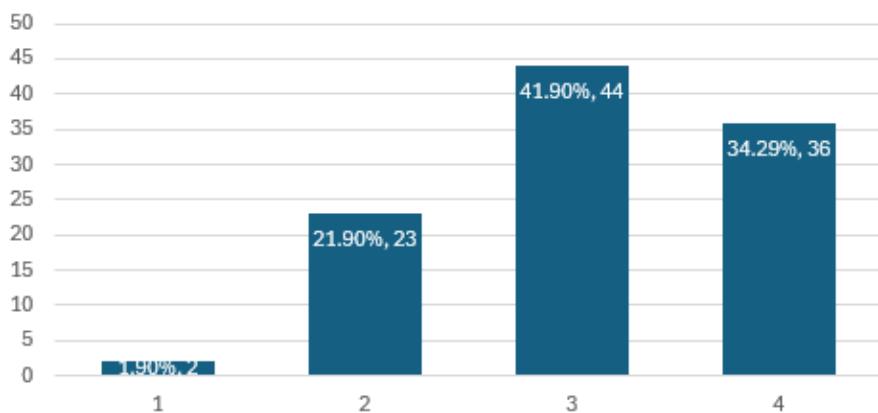


Figure 4. 11 Level of Ordering Problem Impact Production

The results indicate that only a small minority of 1.90% of respondents believe that there is no impact on the product when ordering problems occur. At the same time, 34.29% of respondents hold a completely opposite opinion compared to the 1.90 mentioned earlier. They believe that the ordering problem will cause a significant impact on productivity. Besides that, 41.90% of respondents believed that the ordering problem will cause a moderate impact on productivity. Finally, there are 21.90% of respondents rate for the 2, which is slight impact on the production when ordering problem is happening.

What factors influence your decision to adopt new technologies in your supply chain management?

105 responses

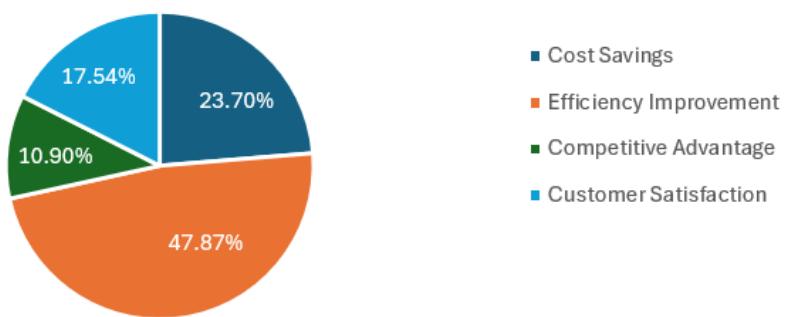


Figure 4. 12 Factors of Using SCMS

This result was collected for getting insights into the factors that could affect the individual in adopting new technologies in their supply chain management. This is a multiple-choice question, as more than one factor could impact motivation in adopting new technologies. Efficiency improvement is the most significant factor at 47.87%. By following, cost saving is also a considerable influence since it has 50 individual votes for this, constituting around 23.70%. Customer satisfaction, with 17.54% of respondents considering it an important factor. The least influential is the competitive advantage, with approximately 11% of votes.

Do you think a supply chain management system can improve your company productivity by improving the ordering process ?

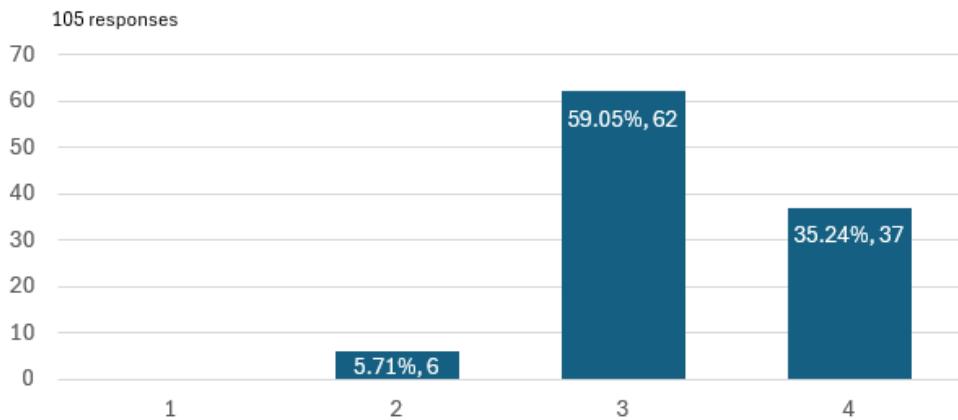


Figure 4. 13 Productivity Improvement of Using Blockchain SCMS

The survey results unveil various levels of trust in the proposed system's potential to enhance company productivity. Approximately 6% of respondents rated their trust at level 2, indicating limited trust. 59.05% of respondents rate their trust level at 3, which is also the biggest portion in this question. Complete trust in the system was expressed by 35.24% of respondents. Lastly, none of the respondents voted for complete lack of trust.

Are you familiar with blockchain technology and its applications in the furniture industry?

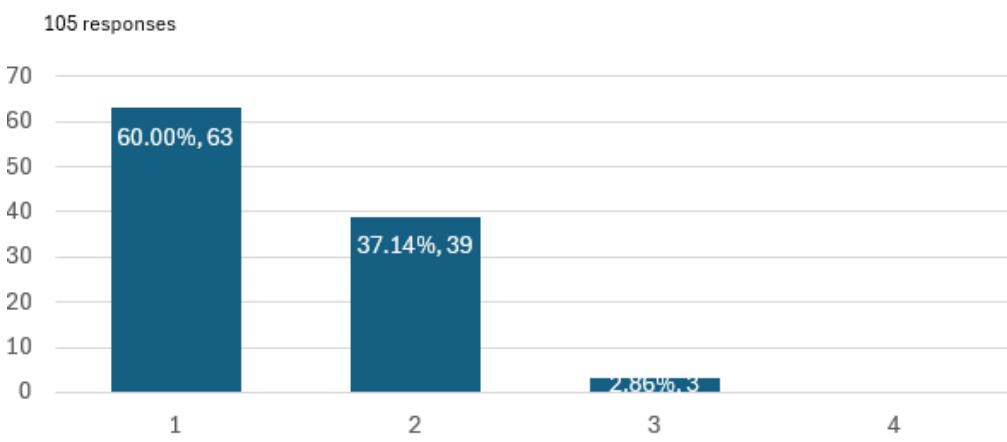


Figure 4. 14 Awareness of Blockchain Technology

The majority of respondents, comprising 60%, claimed that they have no knowledge at all about this technology. A substantial number of respondents, 37.14%, acknowledged having just a basic understanding. Only 2.86% of respondents indicated that they have solid familiarity with blockchain in this industry. Again, none of the participants rated themselves as having good knowledge. Overall, this result indicates that most of the participants are unaware of the blockchain at all.

Do you think the blockchain technology can increase the confidence of each transaction with the supplier and customer?

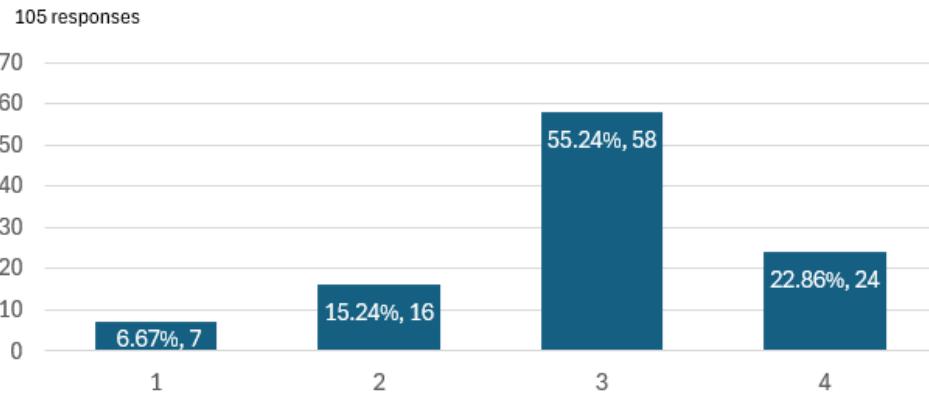


Figure 4. 15 Level of Trust of Using Blockchain SCMS

The survey results unveil various levels of trust in the blockchain can increase transparency of transaction. Approximately 16% of respondents rated their trust at level 2, indicating limited trust. 55.24% of respondents rate their trust level at 3, which is also the biggest portion in this question. Complete trust in the system was expressed by 22.86 of respondents. Lastly, 6.67% of the respondents voted for complete lack of trust.

How willing would you be to adopt this supply chain management system to your business?

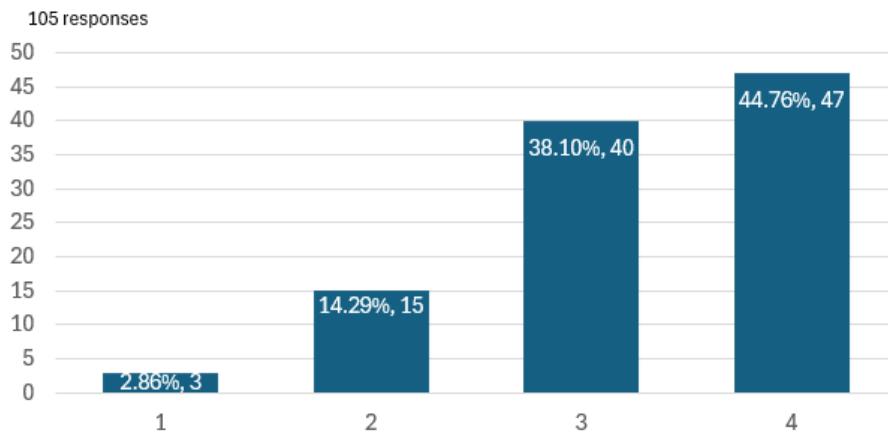


Figure 4. 16 Willingness of Using Blockchain SCMS

Based on 105 responses, a minimal fraction, representing 2.86%, rates their willingness at level 1, indicating a complete lack of willingness. 14.29% of respondents falls under the category of being somewhat willing. A substantial portion, which is around 38% of respondents expressed a willingness to adopt the system. Lastly, the largest segment, comprising 44.76% of respondents, rates their willingness at the highest level in terms of using the proposed system.

4.3.1 Conclusion of Findings

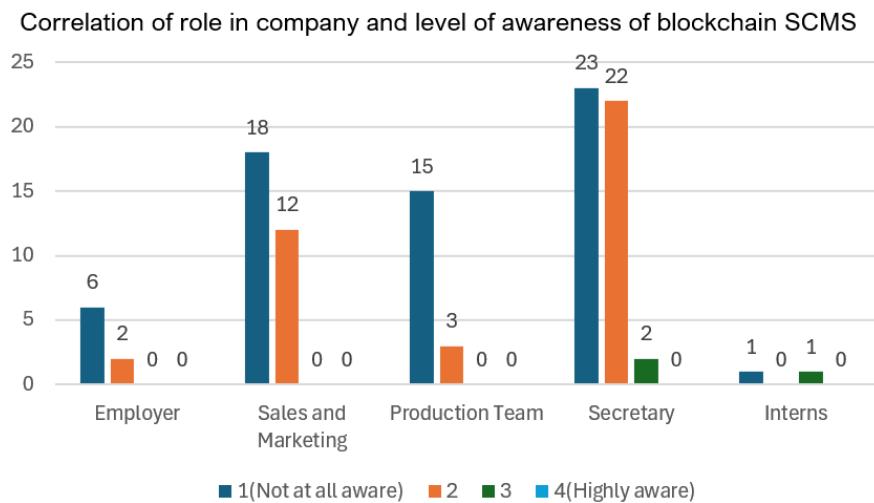


Figure 4. 17 Correlation of Role in Company and Level of Awareness of Blockchain SCMS

The chart shows the distribution of respondents' awareness level of blockchain based on their role in company.

- In the "Not at all aware" category (level 1), individuals in various roles, such as employers, members of the sales and marketing team, production team members, secretaries, or interns, consistently exhibit a complete lack of awareness regarding blockchain technology.
- In the “Slightly aware” category (level 2), consistently representing the second-highest portion across various roles, except interns.
- For the “Moderately aware” category (level 3), there was only 3 respondents vote for it. 2 of them are the secretary in their respective company and the another is the intern.
- For the “Highly aware” category (level 4), no respondents voted for this level.

Overall, most respondents, regardless of their role in the company, fall into the lower awareness levels (Level 1 and Level 2) regarding blockchain technology. There is limited representation in the "Moderately aware" category, with only three respondents, and none of the respondents fall into the "Highly aware" category. This suggests a general lack of high awareness and understanding of blockchain technology across different roles within the surveyed companies.

Correlation of role in company and level of trust of blockchain SCMS

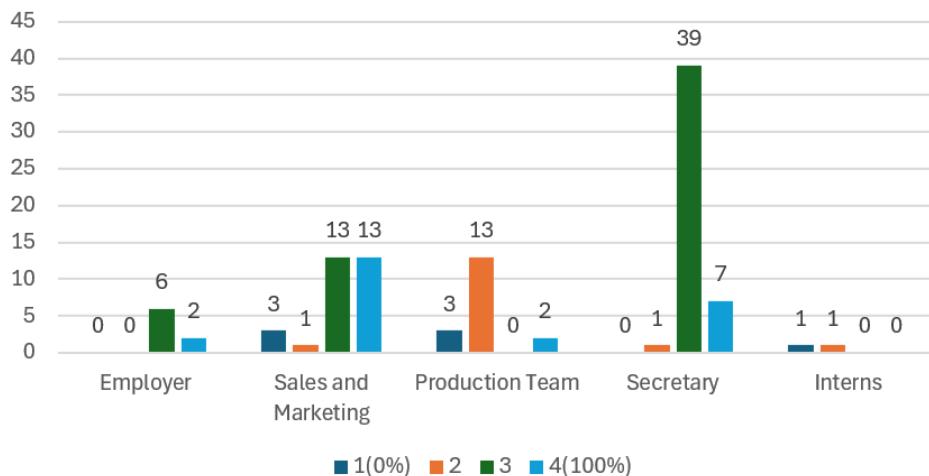


Figure 4. 18 Correlation of Role in Company and Level of Trust of Blockchain SCMS

The chart demonstrates the distribution of respondents' level of trust in blockchain SCMS based on their role in the company.

- For the “1 (0%)” category, which represents respondents who have no trust in blockchain SCMS. Only 7 respondents voted, with 3 from sales and marketing, 3 from the production team, and 1 intern having no trust in blockchain SCMS.
- For the “2” category, representing respondents with slight trust in blockchain SCMS. 16 respondents voted, with the majority (13) from the production team, indicating slight trust. One each from sales and marketing, secretary, and an intern also expressed this level of trust.
- For the “3” category, representing the moderate trust of blockchain SCMS. Most respondents from various roles (employers, sales and marketing, and secretaries) expressed moderate trust in blockchain SCMS (total 58 respondents).
- For the “4 (100%)” category, which represents respondents with full trust in blockchain SCMS. The second-largest segment, with 24 respondents, showed full trust in blockchain SCMS. This group includes employers, sales and marketing, production team members, and secretaries.

Overall, the majority of respondents have some level of trust in blockchain SCMS, with the highest trust level being "Moderate Trust" particularly among employers, sales and marketing team members, and secretaries. Besides that, the distribution across different roles indicates varying levels of trust, with the production team having the highest trust in the "Slight Trust" category. The "Full Trust" category exhibits significant representation, particularly among respondents from the sales and marketing team and secretaries. Conversely, the "No Trust" category, while the smallest, encompasses participants from various roles, suggesting a diversity of perspectives on trust in blockchain SCMS.

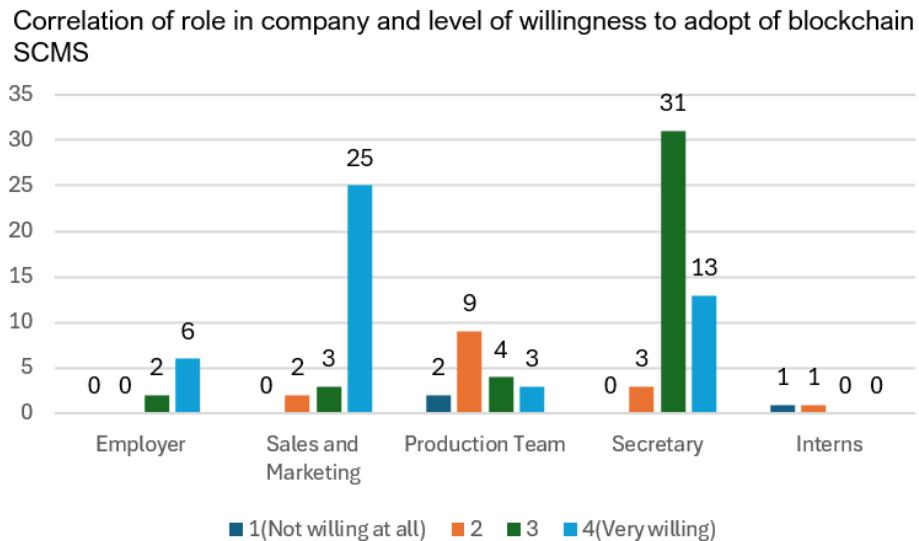


Figure 4. 19 Correlation of Role in Company and Level of Willingness of Using Blockchain SCMS

This chart displays the willingness level of adopting blockchain SCMS based on the role of respondents in their company.

- For the “Not willing at all” category (level 1), which represents respondents who are not willing to adopt the blockchain SCMS. Only 3 respondents voted, with 2 from the production team and 1 intern expressing no willingness to adopt blockchain SCMS.
- For the “Somewhat willing” category (level 2), representing respondents who are somewhat willing to adopt blockchain SCMS. 15 respondents voted, including 2 from sales and marketing, 9 from the production team (the majority), 3 secretaries, and 1 intern expressing a certain degree of willingness.
- For the “Willing” category (level 3), representing the willingness at a certain degree to adopt blockchain SCMS. In total, 40 respondents expressed willingness, with 2 employers, 3 from sales and marketing, 4 from the production team, and 31 secretaries contributing. (No interns voted at this level.)
- For the “Very willing” category (level 4), which represents respondents with the highest level of willingness in using blockchain SCMS. The largest segment with 47 respondents includes employers (6), sales and marketing team members (25), production team members (3), and secretaries (13) showing the highest level of willingness to use blockchain SCMS.

In summary, respondents exhibit diverse attitudes toward adopting the blockchain SCMS. A majority, 47 individuals, express high willingness, particularly from specific roles. The "Willing" category includes 40 participants from various roles (excluding interns), indicating significant openness. Conversely, the "Not willing at all" and "Somewhat willing" categories, with fewer participants, highlight a minority expressing limited willingness or reservations. This nuanced perspective underscores varied responses within the surveyed group.

4.4 Requirements

4.4.1 Use Case Diagram

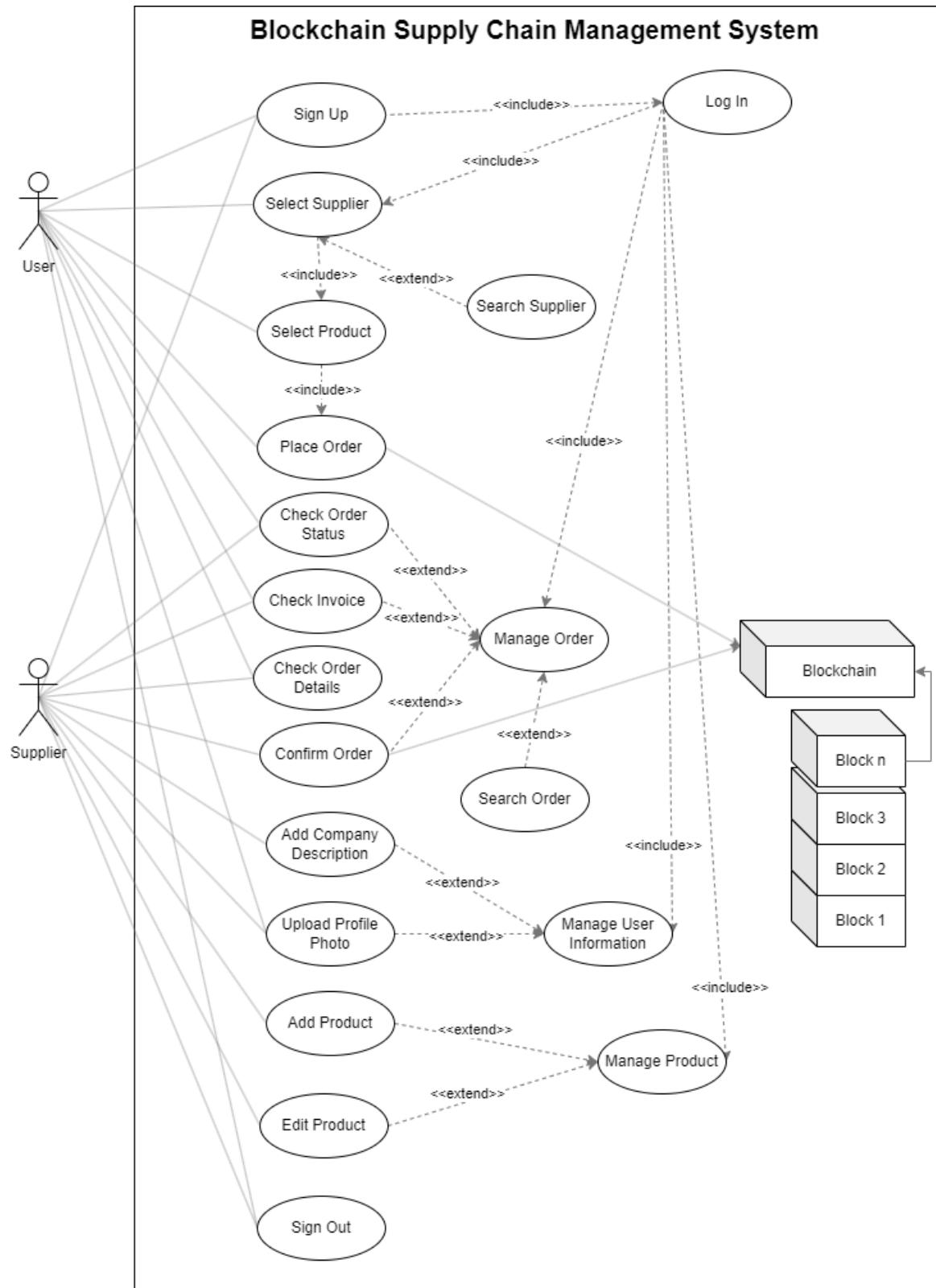


Figure 4. 20 Use Case Diagram – Blockchain SCMS

4.4.2 Use Case Description

UC001 Sign Up

Use Case ID	UC001	Version	1.0
Feature	F001 Sign Up		
Purpose	To allow user or supplier to create an account to access to the system		
Actor	User, Supplier		
Trigger	User or supplier selects “Sign Up” button on the Log In page		
Precondition	<ul style="list-style-type: none"> - User or supplier does not have an account - User or supplier is not logged in - User or supplier is on the Log In page 		
Scenario Name	Step	Action	
Main Flow	1	User or supplier selects “Sign Up” button on Log In page	
	2	System directs user or supplier to Sign Up page	
	3	User or supplier choose the user type	
	4	User or supplier enters company address, company name, email, password	
	5	User or supplier selects “Join now” button	
	6	System validates user or supplier credentials	
	7	System records user credentials to database	
	8	System directs user or supplier to Log In page	
Alternative Flow – Invalid entered user credentials	1.1	User or supplier enters incorrect credentials	
	1.2	System validates user credentials	
	1.3	System displays error message, indicating that the entered credentials are incorrect	
	1.4	Repeat step 4	
Alternative Flow – Account already registered	2.1	User or supplier enters existing email	
	2.2	System validates user credentials	

	2.3	System displays error message, indicating that the account has been registered
	2.4	Repeat step 2
Alternative Flow – Required fill missing	3.1	User or supplier fails to complete all required fields
	3.2	System displays error message, indicating that the required field is not completed
	3.3	Repeat step 3
Rules	<ul style="list-style-type: none"> - Entered credential must be valid and follow required format by the system - All the required fields must be completed - Account does not exist before signing up 	
Author	Goo Han Cong	

Table 4. 4 Use Case Table – Sign Up

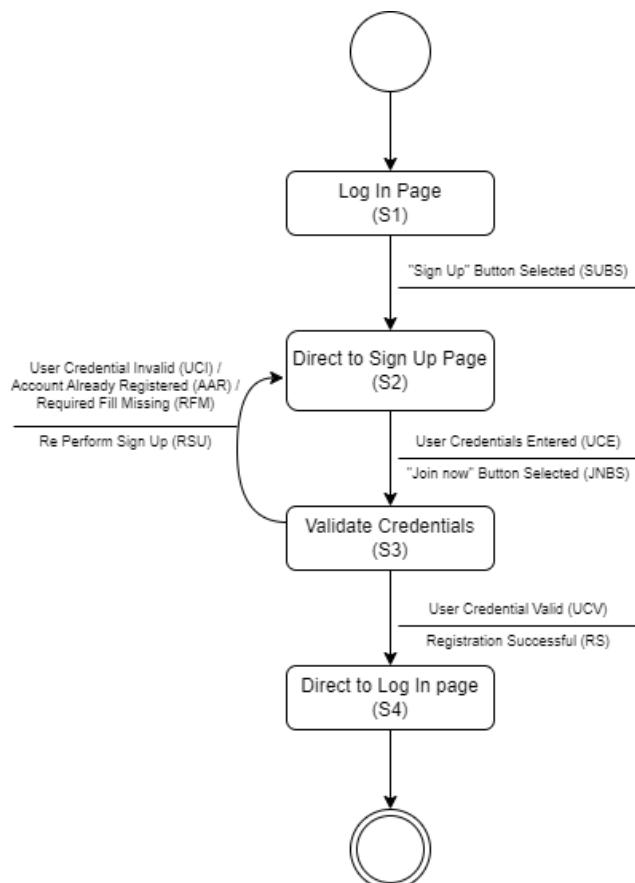


Figure 4. 21 State Chart Diagram – Sign Up

UC002 Log In

Use Case ID	UC002		Version	1.0
Feature	F002 Log In			
Purpose	To allow user or supplier to login and access the system			
Actor	User, Supplier			
Trigger	<ul style="list-style-type: none"> - User or supplier starts the system - User or supplier selects “Log in” button on the Sign Up page 			
Precondition	<ul style="list-style-type: none"> - User or supplier already has an account - User or supplier is not logged in - User or supplier is on the Log In page 			
Scenario Name	Step	Action		
Main Flow	1	User or supplier selects “Log in” button on Sign Up page		
	2	System directs user or supplier to Log In page		
	3	User or supplier enters email and password		
	4	User or supplier selects “Log In” button		
	5	System validates the credentials		
	6	System directs user or supplier to their respective homepage		
Alternative Flow – Invalid entered user credentials	1.1	User or supplier enters incorrect credentials		
	1.2	System validates user credentials		
	1.3	System displays error message, indicating that the entered credentials are invalid		
	1.4	Repeat step 3		
Alternative Flow – Required fill missing	2.1	User or supplier fails to complete all required fields		
	2.2	System displays error message, indicating that the required field is not completed		
	2.3	Repeat step 3		
Rules	<ul style="list-style-type: none"> - Entered credential must be valid and follow required format by the system - All the required fields must be completed 			

	<ul style="list-style-type: none"> - The Email fields must be filled with symbol “@” - Entered credential must be valid
Author	Goo Han Cong

Table 4. 5 Use Case Table – Log In

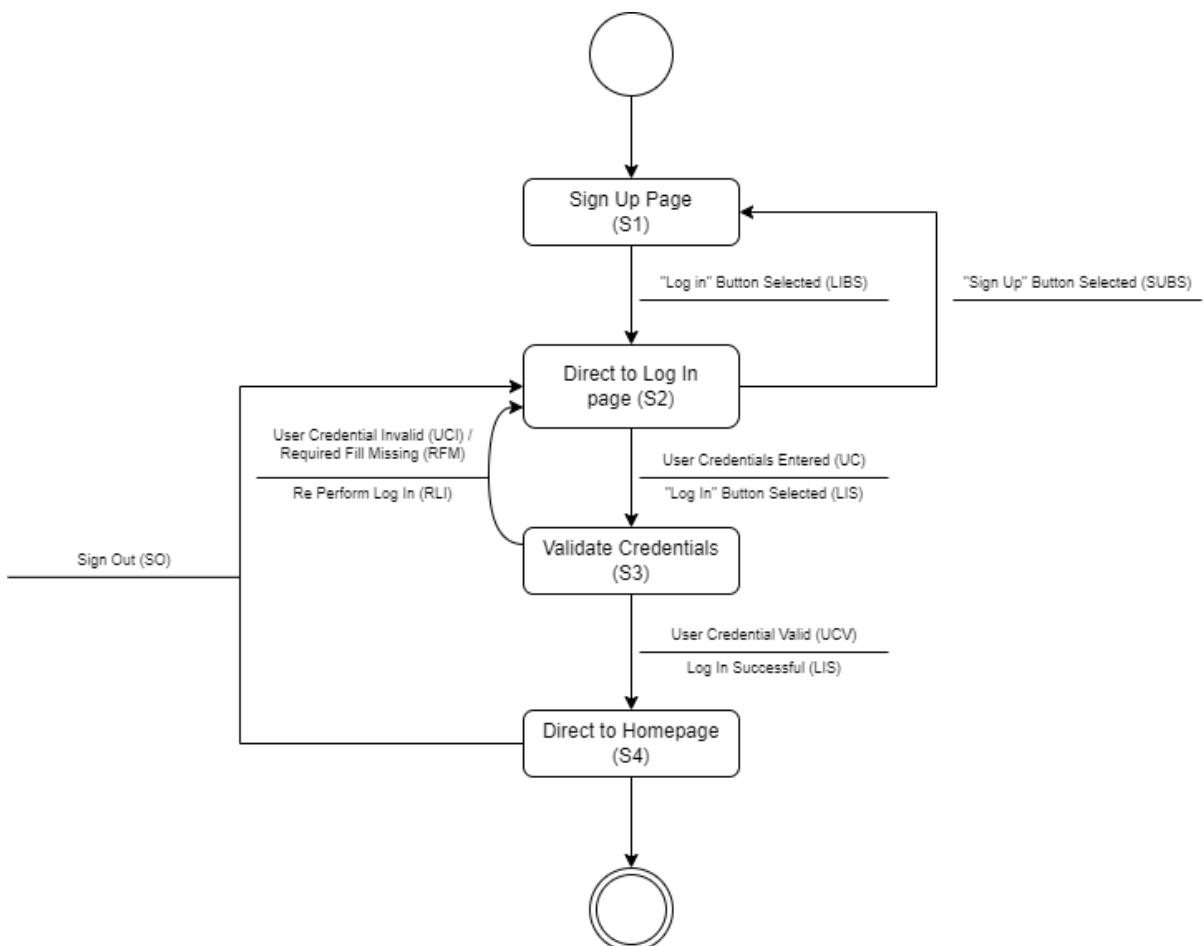


Figure 4. 22 State Chart Diagram – Log In

UC003 Select Supplier

Use Case ID	UC003		Version	1.0			
Feature	F003 Select Supplier						
Purpose	To allow users to select the preferred supplier						
Actor	User						
Trigger	User has logged in and is directed to homepage						
Precondition	<ul style="list-style-type: none"> - User is logged in - User is on the homepage 						
Scenario Name	Step	Action					
Main Flow	1	System displays supplier list on the homepage					
	2	User scrolls for choosing the preferred supplier					
	3	User selects the preferred supplier					
	4	System directs user to Select Product Page					
Alternative Flow	-						
Rules	<ul style="list-style-type: none"> - User must be logged in 						
Author	Goo Han Cong						

Table 4. 6 Use Case Table – Select Supplier

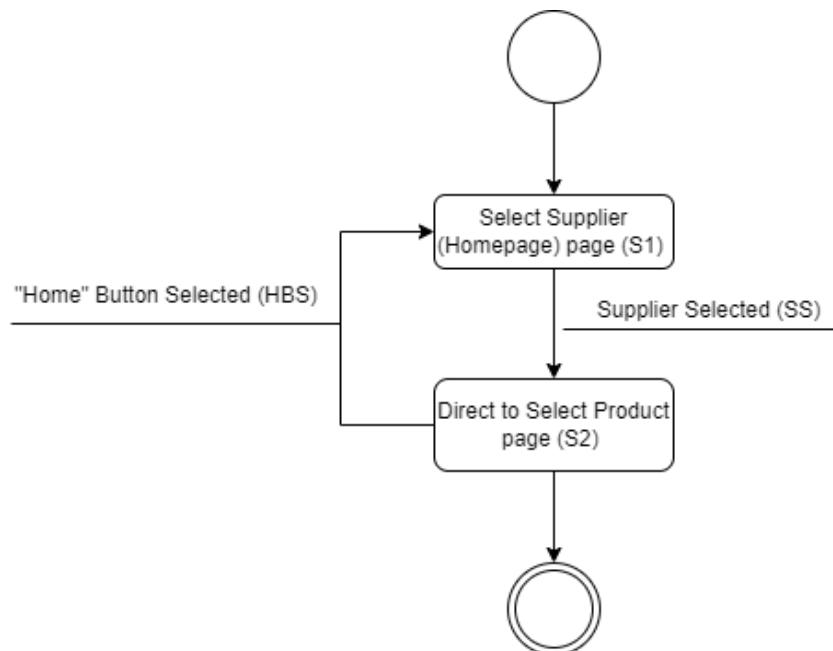


Figure 4. 23 State Chart Diagram – Select Supplier

UC004 Select Product

Use Case ID	UC004		Version	1.0
Feature	F004 Select Product			
Purpose	To allow users to select the preferred product			
Actor	User			
Trigger	User has chosen the preferred supplier on Select Supplier page			
Precondition	<ul style="list-style-type: none"> - User is logged in - User has selected the supplier on Select Supplier page - User is on Select Product page 			
Scenario Name	Step	Action		
Main Flow	1	System displays product list on the Select Product page		
	2	User scrolls the list for choosing the preferred product		
	3	User selects the “+” and “-” buttons to select the preferred products’ quantity		
	4	System displays the order summary (chosen products, quantity, and price) on the left side of interface		
	5	User selects “Proceed to Checkout” button		
	6	System directs to Place Order page		
Alternative Flow – No product is available	1.1	System displays message, indicating that no product is available with the preferred supplier		
Alternative Flow – No product is selected	2.1	User selects “Proceed to Checkout” without selecting any products		
	2.2	System displays error message, indicating that no product is selected		
	2.3	Repeat step 2		
Alternative Flow – Return to Select Supplier Page	3.1	User selects “Home” button or “WoodChain” in the navigation bar		
	3.2	System directs user to the user homepage		

Rules	<ul style="list-style-type: none"> - The supplier must be selected - The product must be selected
Author	Goo Han Cong

Table 4. 7 Use Case Table – Select Product

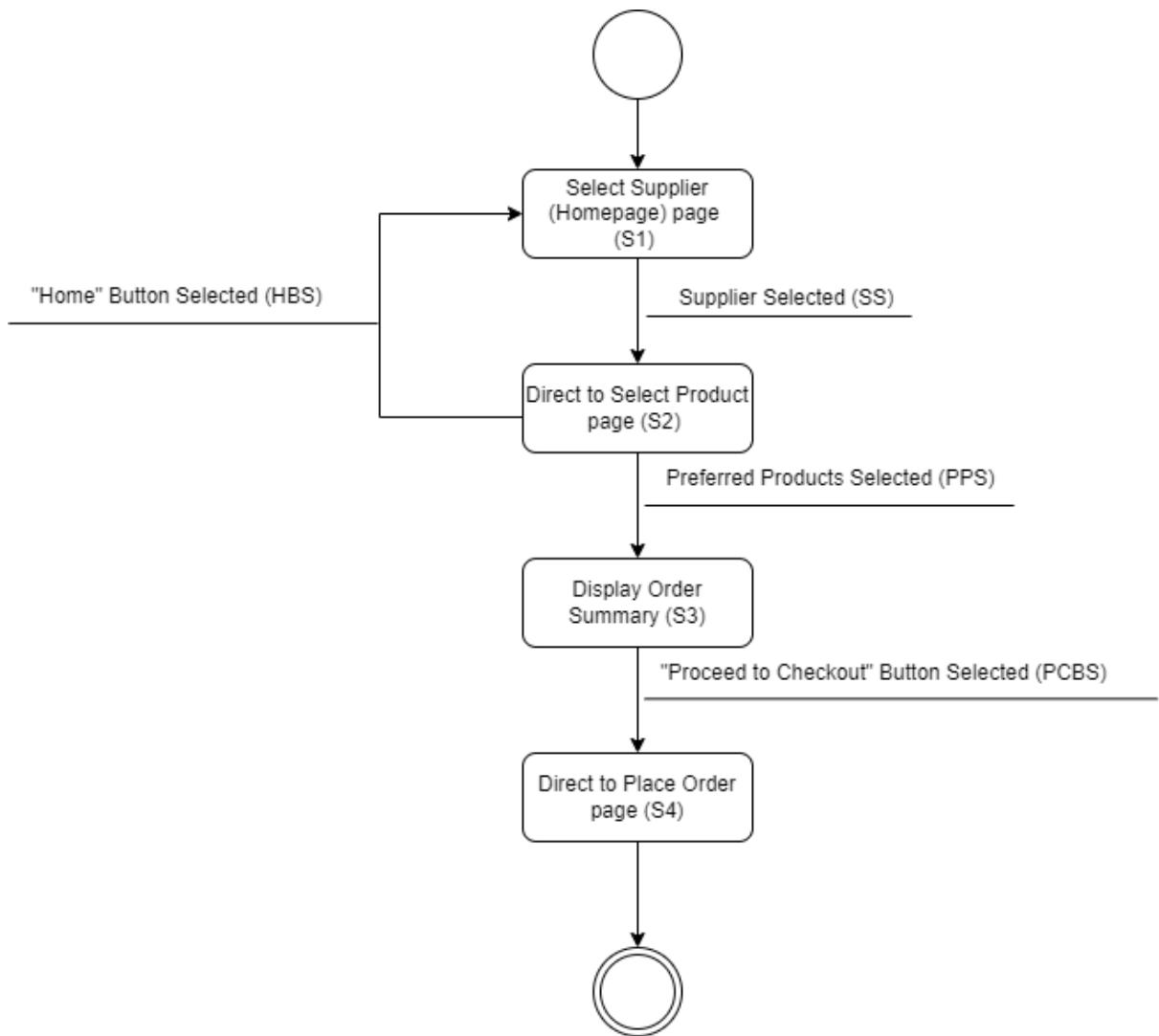


Figure 4. 24 State Chart Diagram – Select Product

UC005 Place Order

Use Case ID	UC005		Version	1.0
Feature	F005 Place Order			
Purpose	To allow users to place the order			
Actor	User			
Trigger	User selects the “Proceed to Checkout” button on the Select Product page			
Precondition	<ul style="list-style-type: none"> - User is logged in - User has chosen the preferred products on Select Product page - User is on Place Order page 			
Scenario Name	Step	Action		
Main Flow	1	System displays order information on the Place Order page		
	2	User selects the intended delivery date		
	3	User checks on the order information		
	4	User selects the “Place Order” button		
	5	System displays Place Order Confirmation modal		
	6	User selects “Confirm” button in Place Order Confirmation modal		
	7	System records order information to database and blockchain		
	8	System directs user to updated Manage Order page		
Alternative Flow – Delivery Date is not selected	1.1	System displays error message, indicating that delivery date is not selected		
	1.2	Repeat step 2		
Alternative Flow – “Cancel” button is selected in confirmation modal	2.1	User selects “Cancel” button in Place Order Confirmation modal		
	2.2	Repeat step 3		
Alternative Flow – Return to Select Supplier Page	3.1	User selects “Home” button or “WoodChain” in the navigation bar		
	3.2	System directs user to the user homepage		
Rules	<ul style="list-style-type: none"> - The delivery date must be selected from date picker 			

	<ul style="list-style-type: none"> - User must need to confirm the action
Author	Goo Han Cong

Table 4. 8 Use Case Table – Place Order

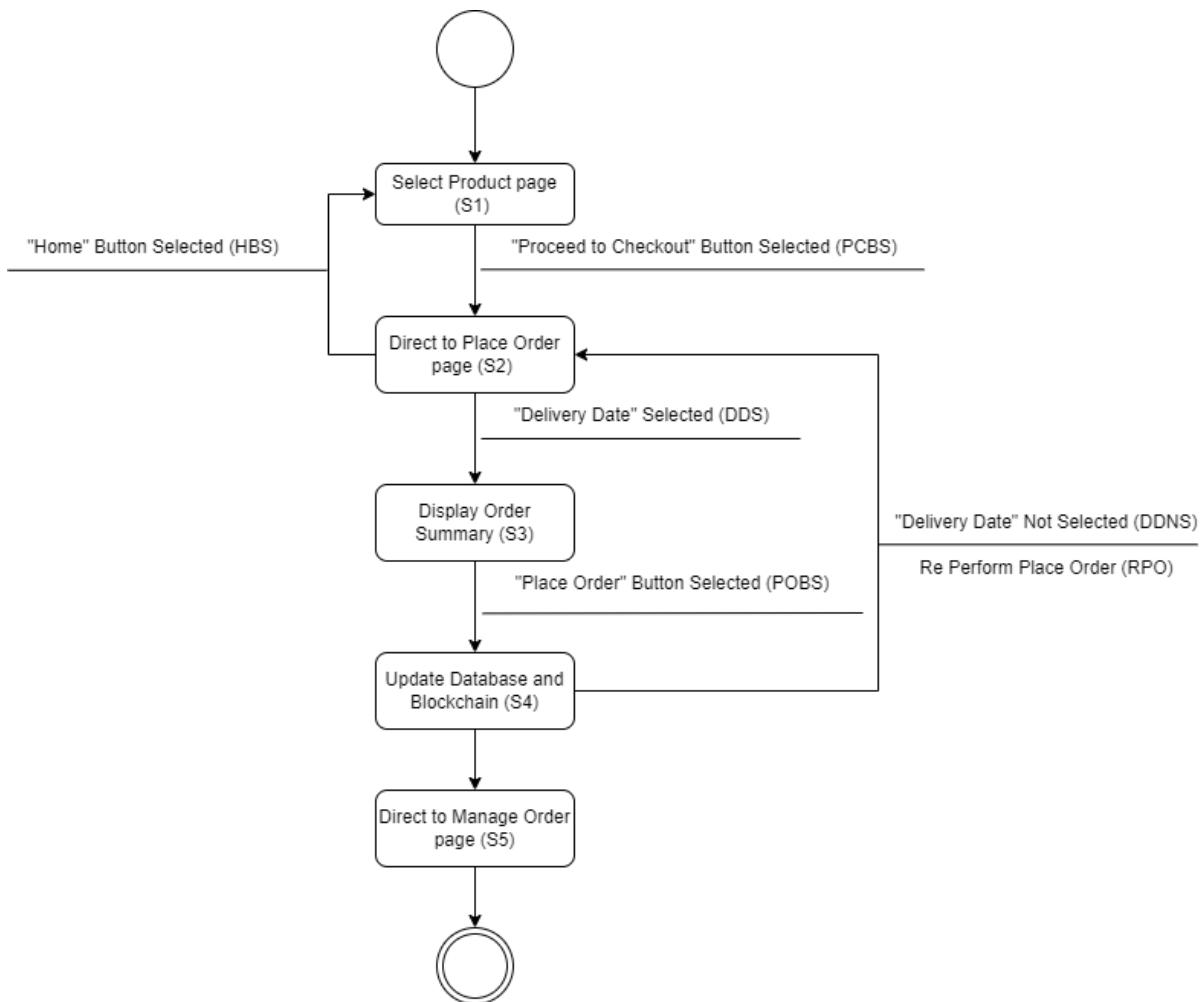


Figure 4. 25 State Chart Diagram – Place Order

UC006 Check Order Status

Use Case ID	UC006		Version	1.0			
Feature	F006 Check Order Status						
Purpose	To allow users or suppliers to check order status						
Actor	User, Supplier						
Trigger	<ul style="list-style-type: none"> - User or supplier selects the “Manage Order” on their respective homepage navigation bar - The “Place Order” Button is selected on Place Order page 						
Precondition	<ul style="list-style-type: none"> - User or supplier is logged in - The order is existing in the list - User or supplier is on Manage Order page 						
Scenario Name	Step	Action					
Main Flow	1	System displays a list of order on the Manage Order page					
	2	User or supplier scrolls the order list					
	3	User or supplier checks the order status					
Alternative Flow – No order is available	1.1	System displays message, indicating that no order is available					
Alternative Flow – Return to homepage	2.1	User or supplier selects “Home” button or “WoodChain” in the navigation bar					
	2.2	System directs user or supplier to their respective homepage					
Rules	<ul style="list-style-type: none"> - User or supplier must be logged in 						
Author	Goo Han Cong						

Table 4. 9 Use Case Table – Check Order Status

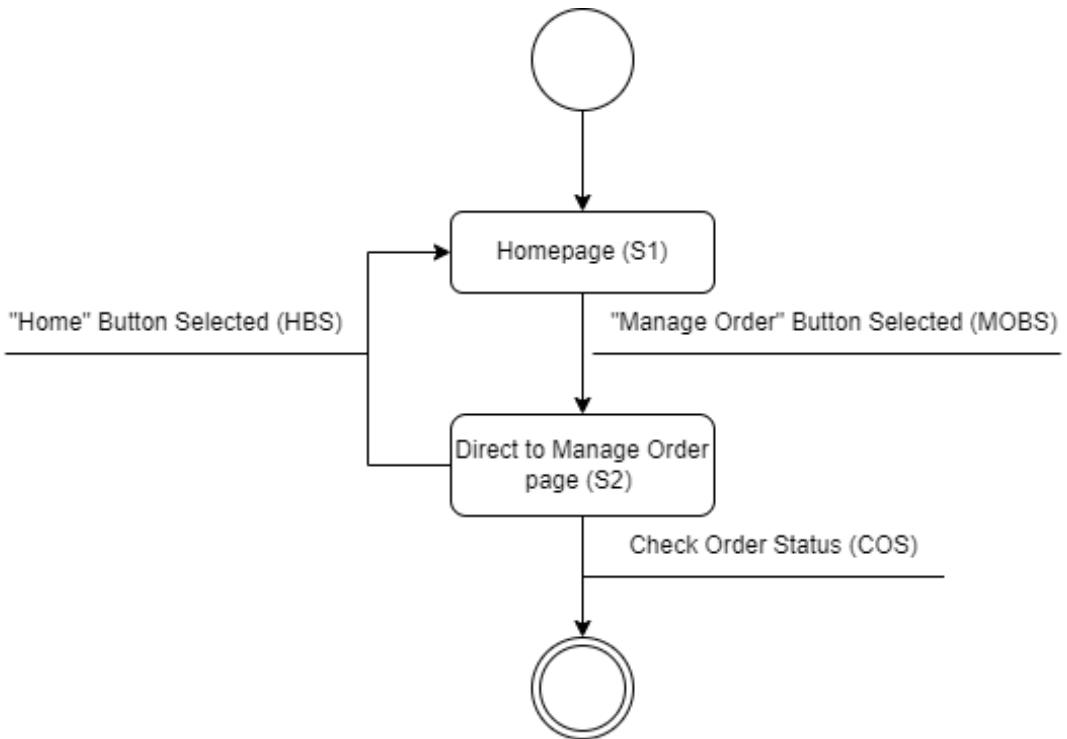


Figure 4. 26 State Chart Diagram – Check Order Status

UC007 Check Invoice

Use Case ID	UC007		Version	1.0			
Feature	F007 Check Invoice						
Purpose	To allow users or suppliers to check the invoice						
Actor	User, Supplier						
Trigger	User or supplier selects “Manage Order” on their respective homepage navigation bar						
Precondition	<ul style="list-style-type: none"> - User or supplier is logged in - The order is existing in the list - The order is confirmed by the supplier - User or supplier is on Manage Order page 						
Scenario Name	Step	Action					
Main Flow	1	System displays a list of order on the Manage Order page					
	2	User or supplier scrolls the order list					
	3	User or supplier selects the order to generate the invoice					
	4	System displays waiting message, indicating that the invoice is being generated					
	5	System displays the PDF invoice based on chosen order					
Alternative Flow – No order is available	1.1	System displays message, indicating that no order is available					
Alternative Flow – Order has not been confirmed	2.1	User or supplier selects the order which has not been confirmed					
	2.2	System displays Order Details modal to demonstrate information based on the chosen order					
Alternative Flow – Return to homepage	3.1	User or supplier selects “Home” button or “WoodChain” in the navigation bar					
	3.2	System directs user or supplier to their respective homepage					
Rules	<ul style="list-style-type: none"> - Supplier must select the order which has been confirmed 						
Author	Goo Han Cong						

Table 4. 10 Use Case Table – Check Invoice

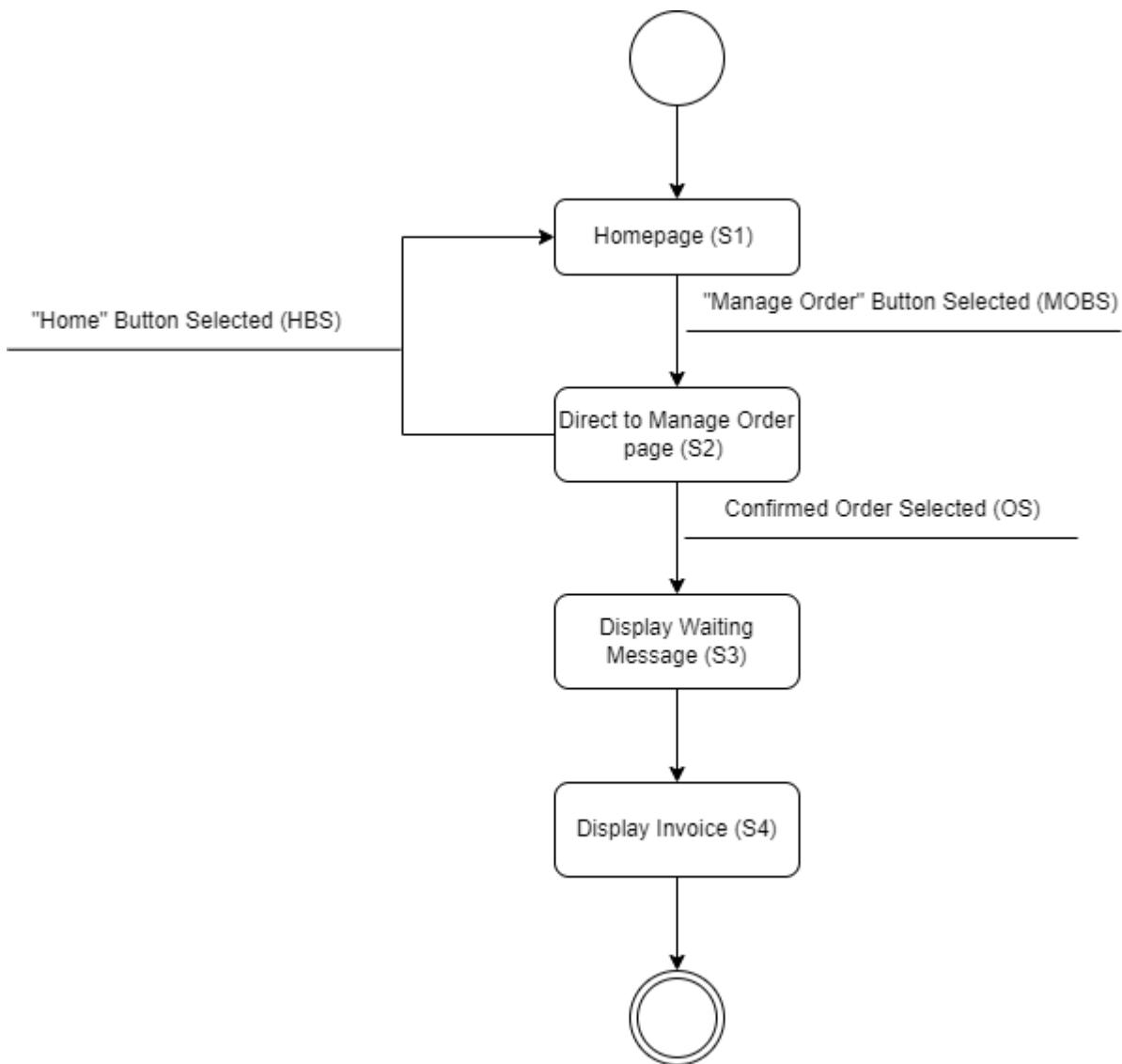


Figure 4. 27 State Chart Diagram – Check Invoice

UC008 Confirm Order

Use Case ID	UC008	Version	1.0
Feature	F008 Confirm Order		
Purpose	To allow suppliers to confirm the order		
Actor	Supplier		
Trigger	Supplier selects “Manage Order” on the supplier homepage		
Precondition	<ul style="list-style-type: none"> - Supplier is logged in - The order is existing in the list - Supplier is on Manage Order page 		
Scenario Name	Step	Action	
Main Flow	1	System displays a list of order on the Manage Order page	
	2	Supplier scrolls the order list	
	3	Supplier checks the order details	
	4	Supplier selects “Confirm Order” button for order	
	5	System displays Confirm Order Confirmation modal	
	6	Supplier selects “Confirm” button in Confirm Order Confirmation modal	
	7	System records order status to database and blockchain	
	8	System displays updated order list	
	9	System displays success message, indicating that the order is confirmed successfully	
Alternative Flow – No order is available	1.1	System displays message, indicating that no order is available	
Alternative Flow – “Cancel” button is selected in confirmation modal	2.1	Suppliers selects “Cancel” button in Confirm Order Confirmation modal	
	2.2	Repeat step 1	
	3.1	Supplier selects “Home” button or “WoodChain” in the navigation bar	

Alternative Flow		
- Return to Manage Product page	3.2	System directs supplier to supplier homepage
Rules	<ul style="list-style-type: none"> - The order must not be confirmed yet - Supplier must need to confirm the action 	
Author	Goo Han Cong	

Table 4. 11 Use Case Table – Confirm Order

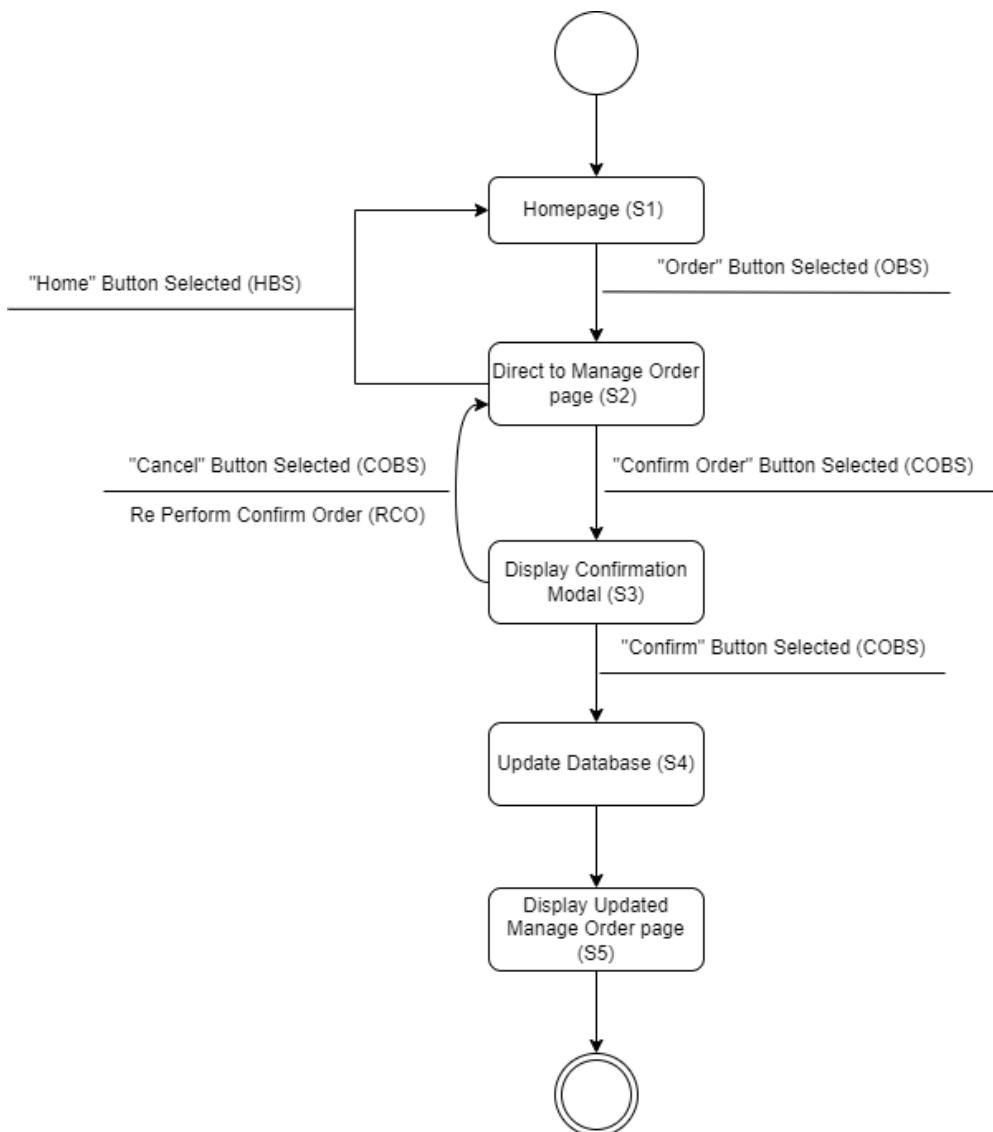


Figure 4. 28 State Chart Diagram – Confirm Order

UC009 Edit Product

Use Case ID	UC009	Version	1.0
Feature	F009 Edit Product		
Purpose	To allow suppliers to update information of product or delete product		
Actor	Supplier		
Trigger	Supplier selects product on the supplier homepage		
Precondition	<ul style="list-style-type: none"> - Supplier is logged in - The product is existing in the list - Supplier is on supplier homepage 		
Scenario Name	Step	Action	
Main Flow	1	System displays a list of product on the supplier homepage	
	2	Supplier scrolls the product list	
	3	Supplier selects on the product which need to be managed	
	4	System displays Manage Product modal	
	5	Supplier updates product information	
	6	Supplier selects “Update Product” Button	
	7	System displays Update Product Confirmation modal	
	8	Supplier selects on “Confirm” button in Update Product Confirmation modal	
	9	System records product information to database	
	10	System displays updated product list	
Alternative Flow – Required field missing	1.1	Supplier fails to complete all required fields	
	1.2	System displays error message, indicating that the required field is not completed	
	1.3	Repeat step 4	
Alternative Flow - Delete product	2.1	Supplier selects “Delete Product” button in Manage Product modal	
	2.2	System displays Delete Product Confirmation modal	

	2.3	Supplier selects on “Delete” button in Delete Product Confirmation modal
	2.4	System records product information to database
	2.5	System displays updated product list
Alternative Flow - “Cancel” button is selected in confirmation modal	3.1	Suppliers selects “Cancel” in Update or Delete Product Confirmation modal
	3.2	Repeat step 4
Alternative Flow - Manage Product modal is closed	4.1	Supplier closes the Manage Product modal
	4.2	Repeat step 1
Rules	<ul style="list-style-type: none"> - When updating product information, - all the required fields must be completed, except for the product photo - the price field must be filled with a number to two decimal places - Supplier must need to confirm the action 	
Author	Goo Han Cong	

Table 4. 12 Use Case Table – Edit Product

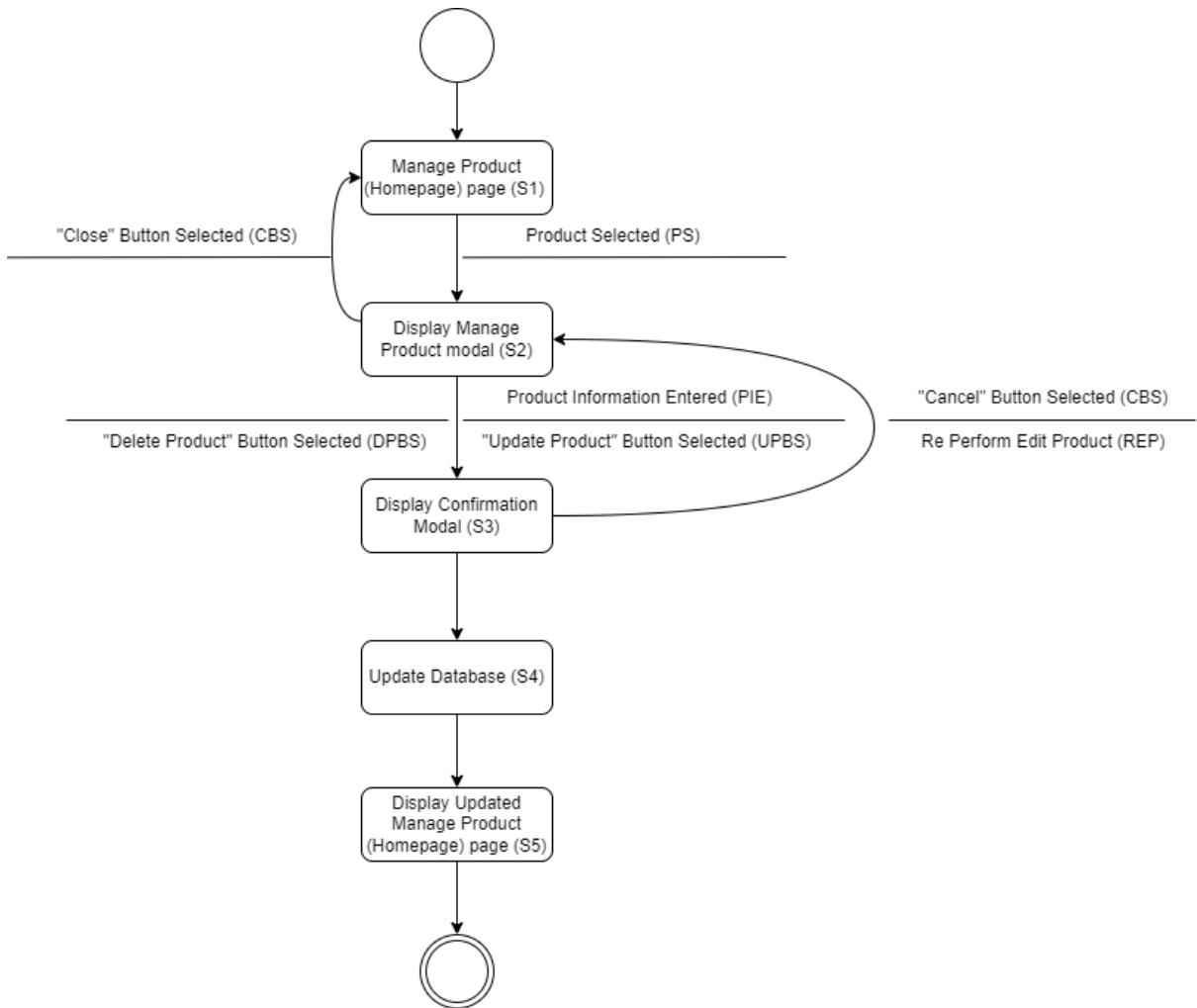


Figure 4. 29 State Chart Diagram – Edit Product

UC010 Add Product

Use Case ID	UC010		Version	1.0
Feature	F010 Add Product			
Purpose	To allow suppliers to add product to product list			
Actor	Supplier			
Trigger	Supplier selects “Add Product” button on the supplier homepage			
Precondition	<ul style="list-style-type: none"> - Supplier is logged in - Supplier is on supplier homepage 			
Scenario Name	Step	Action		
Main Flow	1	System displays a list of product on the supplier homepage		
	2	Supplier selects “Add Product” button		
	3	System displays Add Product modal		
	4	Supplier enters product information		
	5	Supplier selects “Add Product” button on Add Product modal		
	6	Systems displays Add Product Confirmation modal		
	7	Supplier selects “Confirm” button on Add Product Confirmation modal		
	8	System record product information to database		
	9	System displays updated product list		
Alternative Flow – Required field missing	1.1	Supplier fails to complete all required fields		
	1.2	System displays error message, indicating that the required field is not completed		
	1.3	Repeat step 4		
Alternative Flow - “Cancel” button is selected in confirmation modal	2.1	Supplier selects “Cancel” button on Add Product Confirmation modal		
	2.2	Repeat step 3		
Alternative Flow	3.1	Supplier selects “Cancel” button on Add Product modal		

- Add Product modal is closed	3.2	Repeat step 1
Rules		<ul style="list-style-type: none"> - All the required fields must be completed, except for the product photo - The price field must be filled with a number to two decimal places - Supplier must need to confirm the action
Author		Goo Han Cong

Table 4. 13 Use Case Table – Add Product

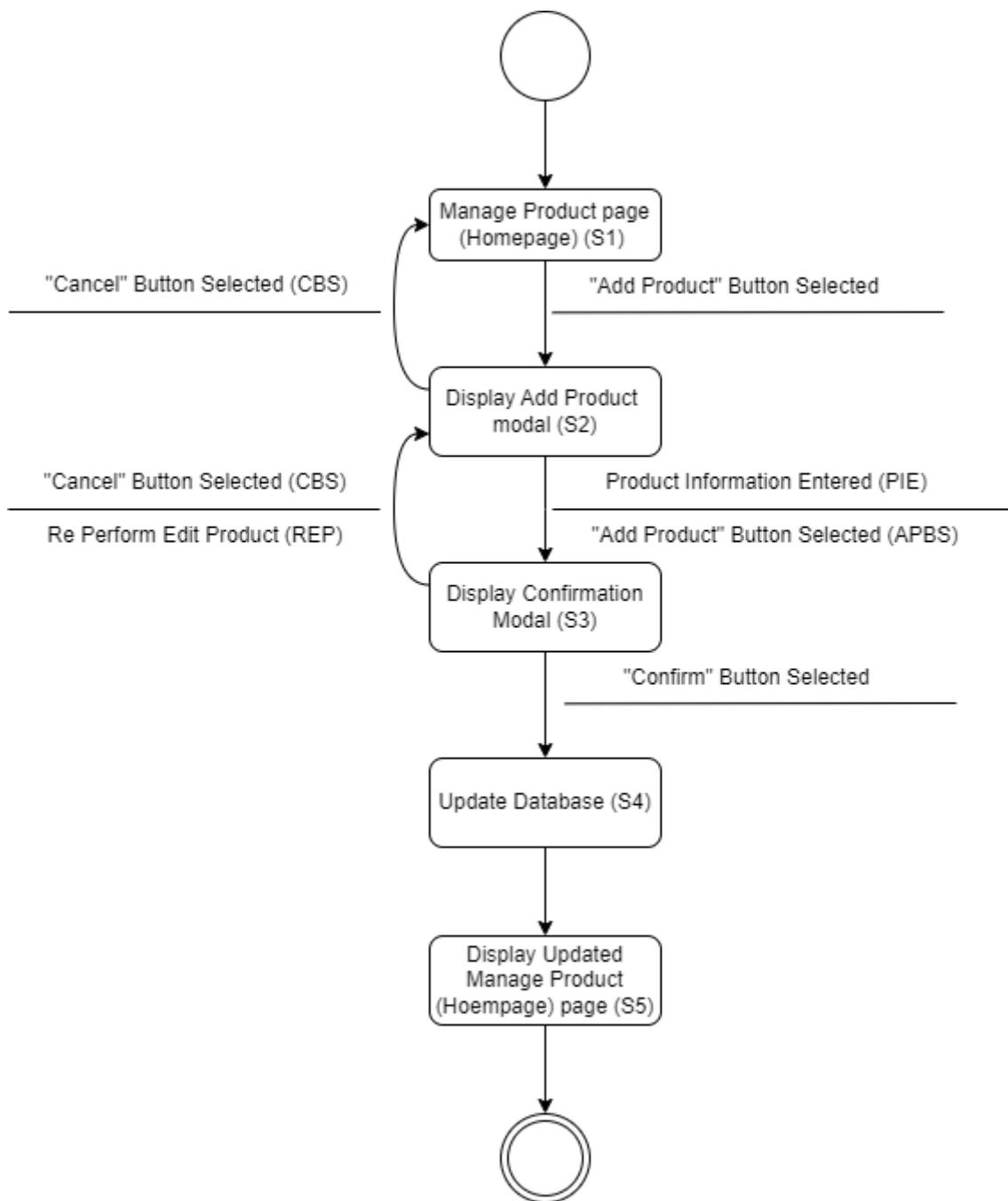


Figure 4. 30 State Chart Diagram – Add Product

UC011 Check Order Details

Use Case ID	UC011	Version	1.0
Feature	F011 Check Order Details		
Purpose	To allow users or suppliers to check order details before order has been confirmed		
Actor	User, Supplier		
Trigger	User or supplier selects the “Manage Order” on their respective homepage navigation bar		
Precondition	<ul style="list-style-type: none"> - User or supplier is logged in - The order is existing in the list - User or supplier is on Manage Order page 		
Scenario Name	Step	Action	
Main Flow	1	System displays a list of order on the Manage Order page	
	2	User or supplier scrolls the order list	
	3	User or supplier selects the order to check order details	
	4	System displays Order Details modal	
	5	User or supplier checks the order details	
	6	Systems displays updated user or supplier list	
Alternative Flow - No order is available	1.1	System displays message, indicating that no order is available	
Alternative Flow - Order has been confirmed	2.1	User or supplier selects the order which has been confirmed	
	2.2	System displays waiting message, indicating that the invoice is being generated	
	2.3	System displays the PDF invoice based on chosen order	
Alternative Flow - Return to homepage	3.1	User or supplier selects “Home” button or “WoodChain” in the navigation bar	
	3.2	System directs user or supplier to their respective homepage	
Rules	<ul style="list-style-type: none"> - User or supplier must select the order which has not been confirmed 		

Author	Goo Han Cong
---------------	--------------

Table 4. 14 Use Case Table – Check Order Details

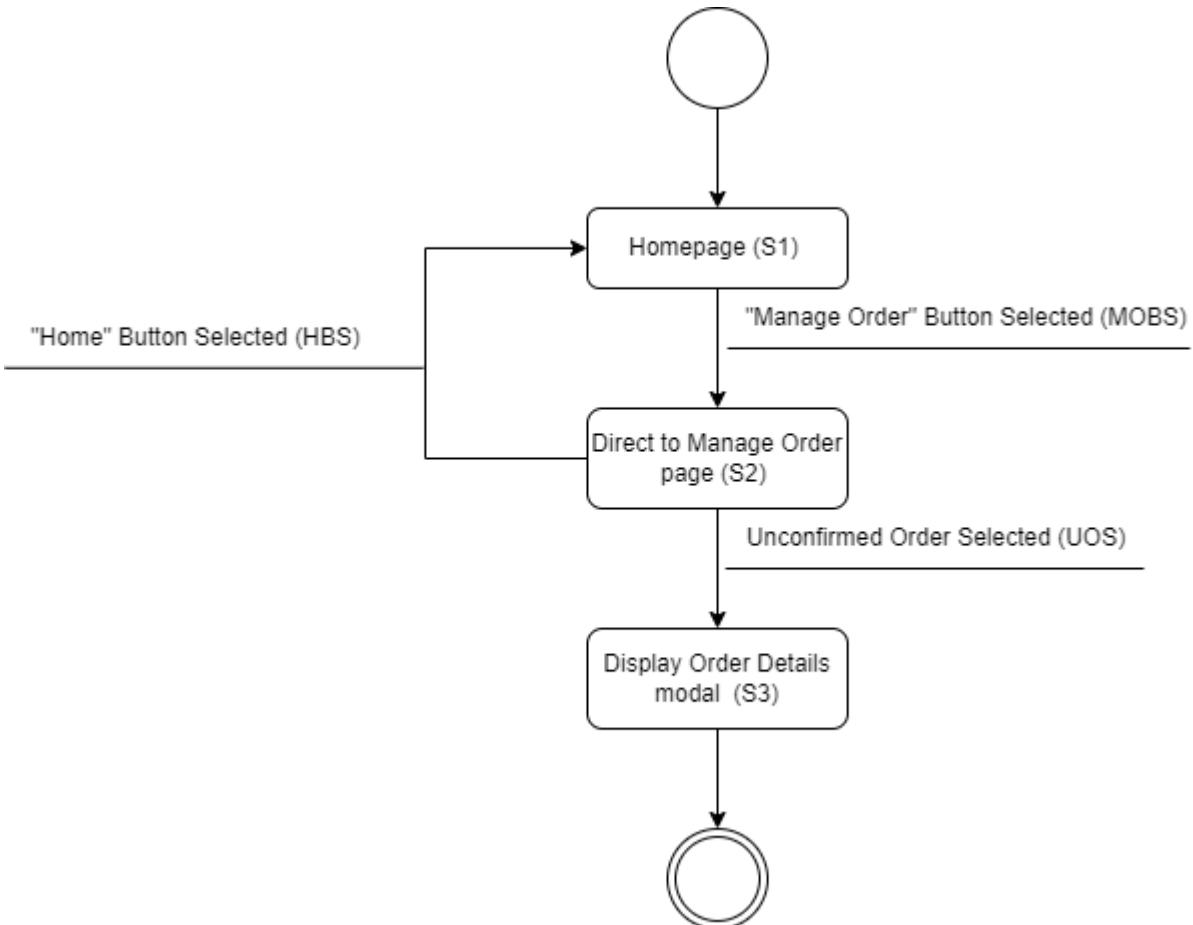


Figure 4. 31 State Chart Diagram – Check Order Details

UC012 Add Company Description

Use Case ID	UC012	Version	1.0		
Feature	F012 Add Company Description				
Purpose	To allow suppliers to update company description				
Actor	Supplier				
Trigger	Supplier selects the profile icon in navigation bar				
Precondition	<ul style="list-style-type: none"> - Supplier is logged in - Supplier is on the page where profile icon is available 				
Scenario Name	Step	Action			
Main Flow	1	Supplier selects the profile icon in navigation bar			
	2	System displays Profile Options modal			
	3	Supplier selects “Add Company Description” button in Profile Options modal			
	4	System displays Add Company Description modal			
	5	Supplier enters company description			
	6	Supplier selects “Save Description” button in Profile Options modal			
	7	System records updated description to database			
	8	System displays the page where the supplier on again			
Alternative Flow - Add Company Description modal is closed	1.1	Supplier selects “Close” button in the Add Company Description modal			
	1.2	Repeat step 8			
Alternative Flow - Profile Options modal is closed	2.1	Supplier closes the Profile Options modal			
	2.2	Repeat step 1			
Rules	<ul style="list-style-type: none"> - Supplier must need to confirm the action 				
Author	Goo Han Cong				

Table 4. 15 Use Case Table – Add Company Description

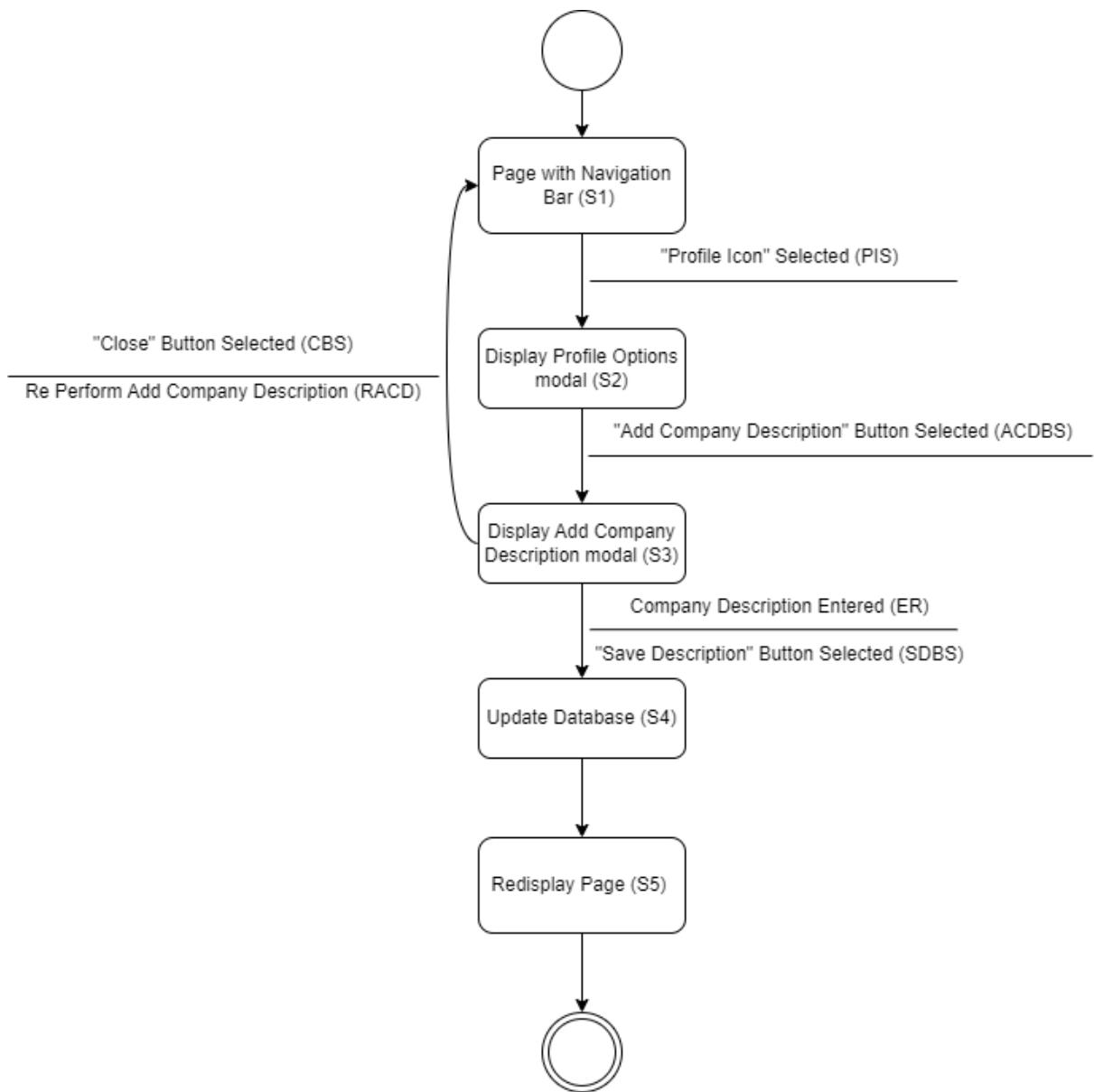


Figure 4. 32 State Chart Diagram – Add Company Description

UC013 Upload Profile Photo

Use Case ID	UC013		Version	1.0
Feature	F013 Upload Profile Photo			
Purpose	To allow users or suppliers to upload profile photo			
Actor	User, Supplier			
Trigger	User or supplier selects the profile icon in navigation bar			
Precondition	<ul style="list-style-type: none"> - User or supplier is logged in - User or supplier is on the page where profile icon is available 			
Scenario Name	Step	Action		
Main Flow	1	Supplier selects the profile icon in navigation bar		
	2	System displays Profile Options modal		
	3	User or supplier selects “Upload Photo” button in Profile Options modal		
	4	System displays Upload Profile Photo modal		
	5	User or supplier selects “Choose File” field		
	6	System directs user or supplier to file explorer		
	7	User or supplier selects preferred photo		
	8	User or supplier selects “Upload” button in Upload Profile Photo modal		
	9	System record profile photo pointer to database		
	10	System displays the page where the user or supplier on again with updated profile photo		
Alternative Flow - Upload Profile Photo modal is closed	1.1	Supplier selects “Close” button in the Add Company Description modal		
	1.2	Repeat step 10		
Alternative Flow - Profile Options modal is closed	2.1	Supplier closes the Profile Options modal		
	2.2	Repeat step 1		
Rules	<ul style="list-style-type: none"> - User or supplier must need to confirm the action - Only file in PNG, JPG, or JPEG format is selectable 			

Author	Goo Han Cong
---------------	--------------

Table 4. 16 Use Case Table – Upload Profile Photo

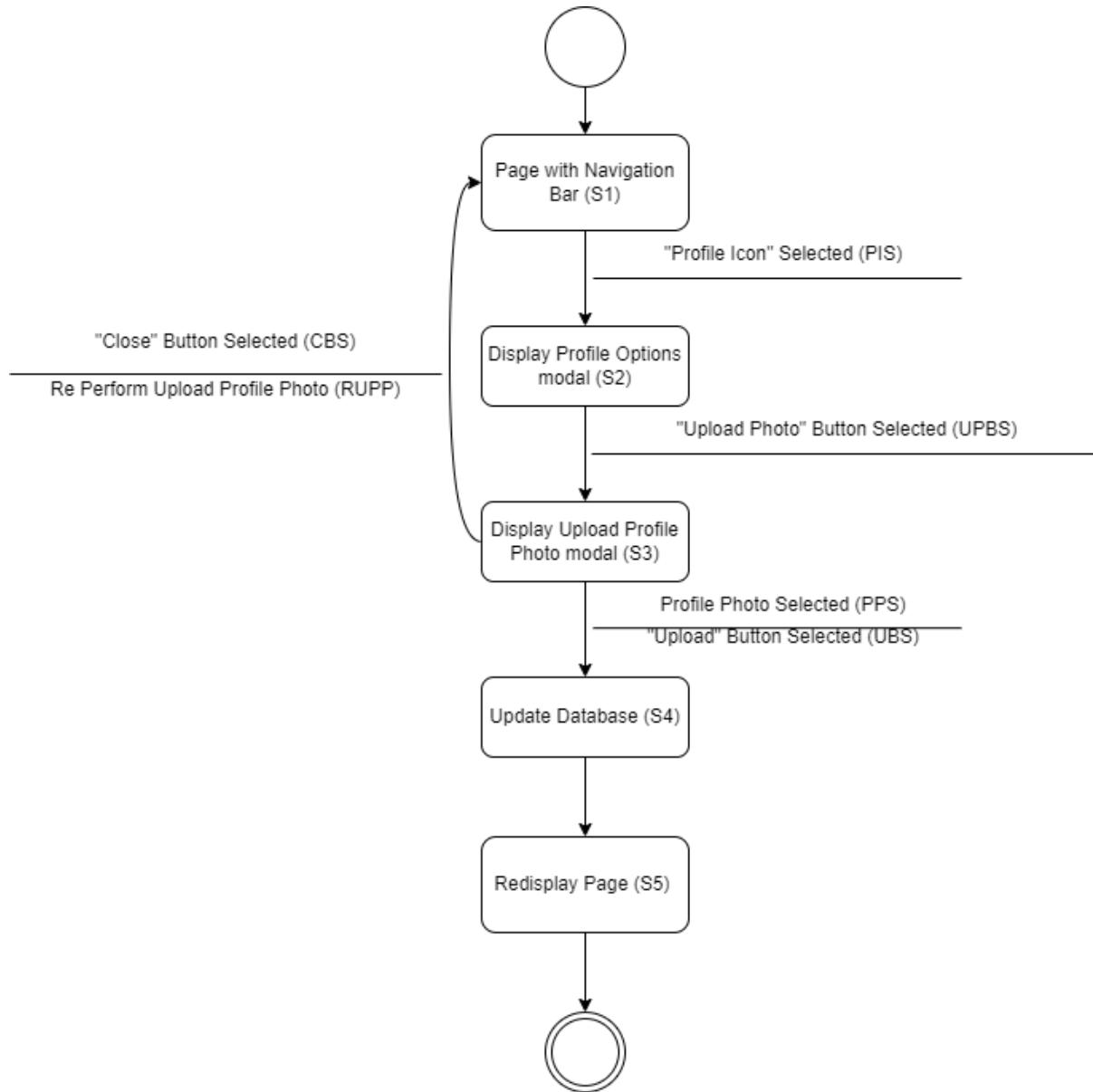


Figure 4. 33 State Chart Diagram – Upload Profile Photo

UC014 Search Supplier

Use Case ID	UC014		Version	1.0			
Feature	F014 Search Supplier						
Purpose	To allow users to search the preferred supplier						
Actor	User						
Trigger	User wants to search the preferred supplier when selecting supplier						
Precondition	<ul style="list-style-type: none"> - User or supplier is logged in - User is on the homepage 						
Scenario Name	Step	Action					
Main Flow	1	User enters the supplier company name for searching supplier in search bar					
	2	User presses on “Enter” on keyboard or select “Search” button on user homepage					
	3	System displays the supplier which has been search					
Alternative Flow - No supplier is found	1.1	System displays message, indicating that no supplier is found					
	1.2	User clears the search bar					
	1.3	Repeat step 1					
Rules	<ul style="list-style-type: none"> - User must need to search existing supplier - User must need to presses on “Enter” on keyboard or select “Search” on the homepage 						
Author	Goo Han Cong						

Table 4. 17 Use Case Table – Search Supplier

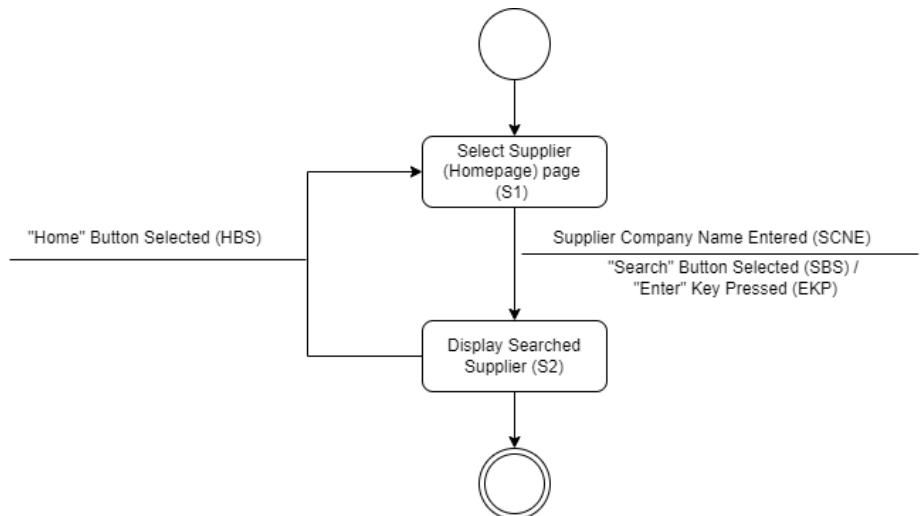


Figure 4. 34 State Chart Diagram – Search Supplier

UC015 Search Order

Use Case ID	UC015		Version	1.0			
Feature	F015 Search Order						
Purpose	To allow users and suppliers to search the order they want to check						
Actor	User, Supplier						
Trigger	User or Supplier wants to search the order for checking						
Precondition	<ul style="list-style-type: none"> - User or supplier is logged in - User or supplier is on the Manage Order page 						
Scenario Name	Step	Action					
Main Flow	1	User or supplier enters the supplier company name or user company name for searching order in search bar					
	2	User or supplier presses on “Enter” on keyboard or select “Search” button on their Manage Order page					
	3	System displays the order which has been search					
Alternative Flow - No order is found	1.1	System displays message, indicating that no order is found					
	1.2	User clears the search bar					
	1.3	Repeat step 1					
Alternative Flow – Return to homepage	2.1	User selects “Home” button or “WoodChain” in the navigation bar					
	2.2	System directs user to the user homepage					
Rules	<ul style="list-style-type: none"> - User and supplier must need to search existing order - User and supplier must need to press on “Enter” on keyboard or select “Search” on the homepage 						
Author	Goo Han Cong						

Table 4. 18 Use Case Table – Search Order

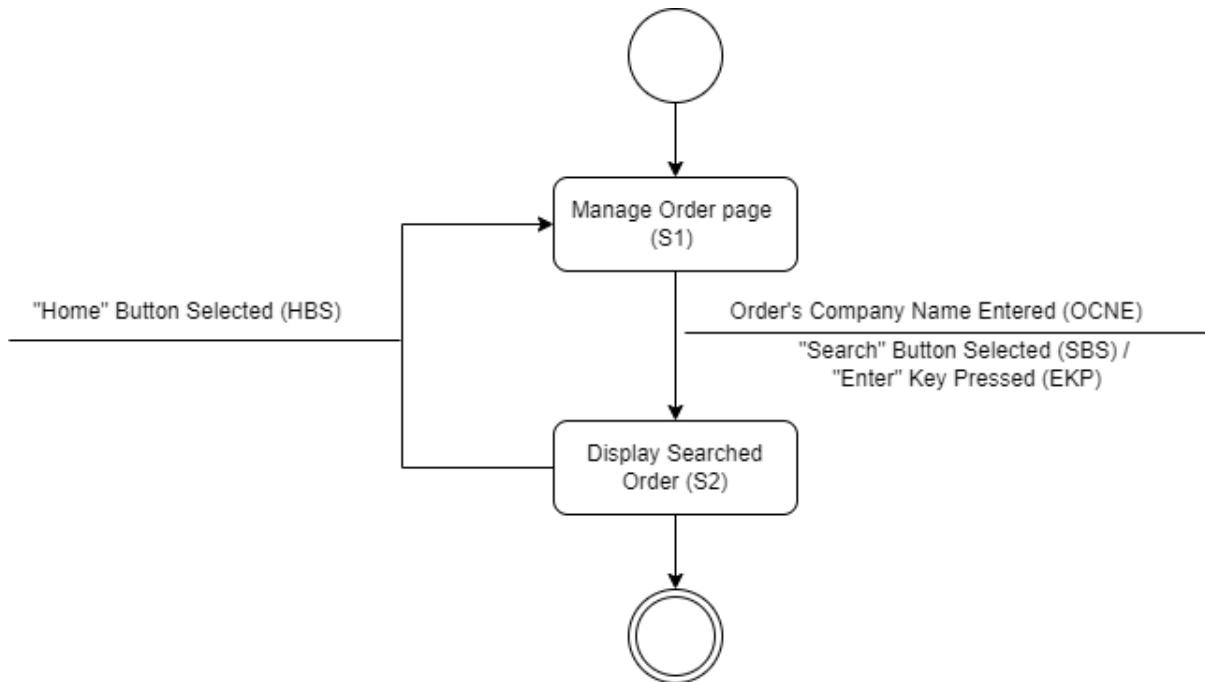


Figure 4. 35 State Chart Diagram – Search Order

UC016 Search Product

Use Case ID	UC016	Version	1.0		
Feature	F016 Search Product				
Purpose	To allow suppliers to search the order they want to manage				
Actor	Supplier				
Trigger	Supplier wants to search the order for managing				
Precondition	<ul style="list-style-type: none"> - Supplier is logged in - Supplier is on the homepage 				
Scenario Name	Step	Action			
Main Flow	1	Supplier enters the product name for searching product in search bar			
	2	Supplier presses on “Enter” on keyboard or select “Search” button on the homepage			
	3	System displays the product which has been search			
Alternative Flow - No product is found	1.1	System displays message, indicating that no product is found			
	1.2	User clears the search bar			
	1.3	Repeat step 1			
Rules	<ul style="list-style-type: none"> - Supplier must need to search existing order - Supplier must need to press on “Enter” on keyboard or select “Search” on the homepage 				
Author	Goo Han Cong				

Table 4. 19 Use Case Table – Search Product

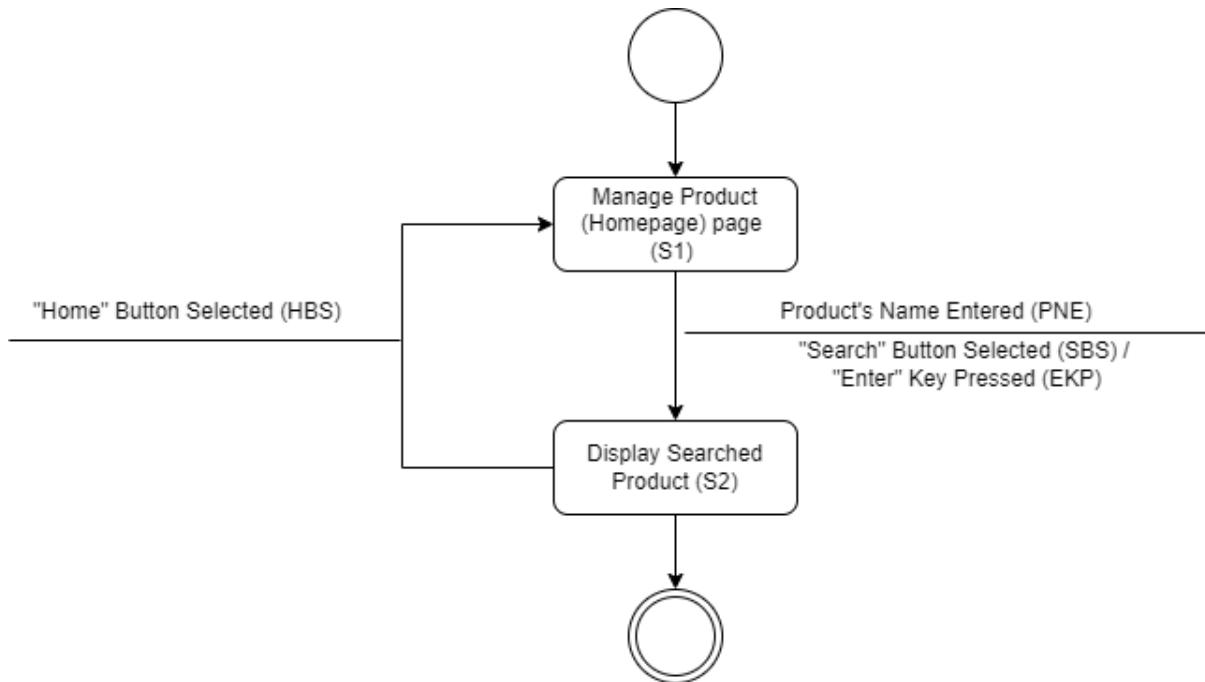


Figure 4. 36 State Chart Diagram – Search Product

UC017 Sign Out

Use Case ID	UC017		Version	1.0			
Feature	F017 Sign Out						
Purpose	To allow users or suppliers to sign out						
Actor	User, Supplier						
Trigger	User or supplier selects the profile icon in navigation bar						
Precondition	<ul style="list-style-type: none"> - Supplier is logged in - Supplier is on the page where profile icon is available 						
Scenario Name	Step	Action					
Main Flow	1	User or supplier selects profile icon in navigation bar					
	2	System displays Profile Options modal					
	3	User or suppliers select “Sign Out” button in Profile Options modal					
	4	System displays Sign Out Confirmation modal					
	5	User or supplier selects “Sign Out” button in Sign Out Confirmation modal					
	6	System directs user or supplier to Log In page					
Alternative Flow - Sign Out Confirmation modal is closed	1.1	User or supplier selects “Cancel” button in Sign Out Confirmation modal					
	1.2	Repeat step 1					
Alternative Flow - Profile Options modal is closed	2.1	User or supplier closes the Profile Options modal					
	2.2	Repeat step 1					
Rules	<ul style="list-style-type: none"> - User or supplier must need to confirm the action 						
Author	Goo Han Cong						

Table 4. 20 Use Case Table – Sign Out

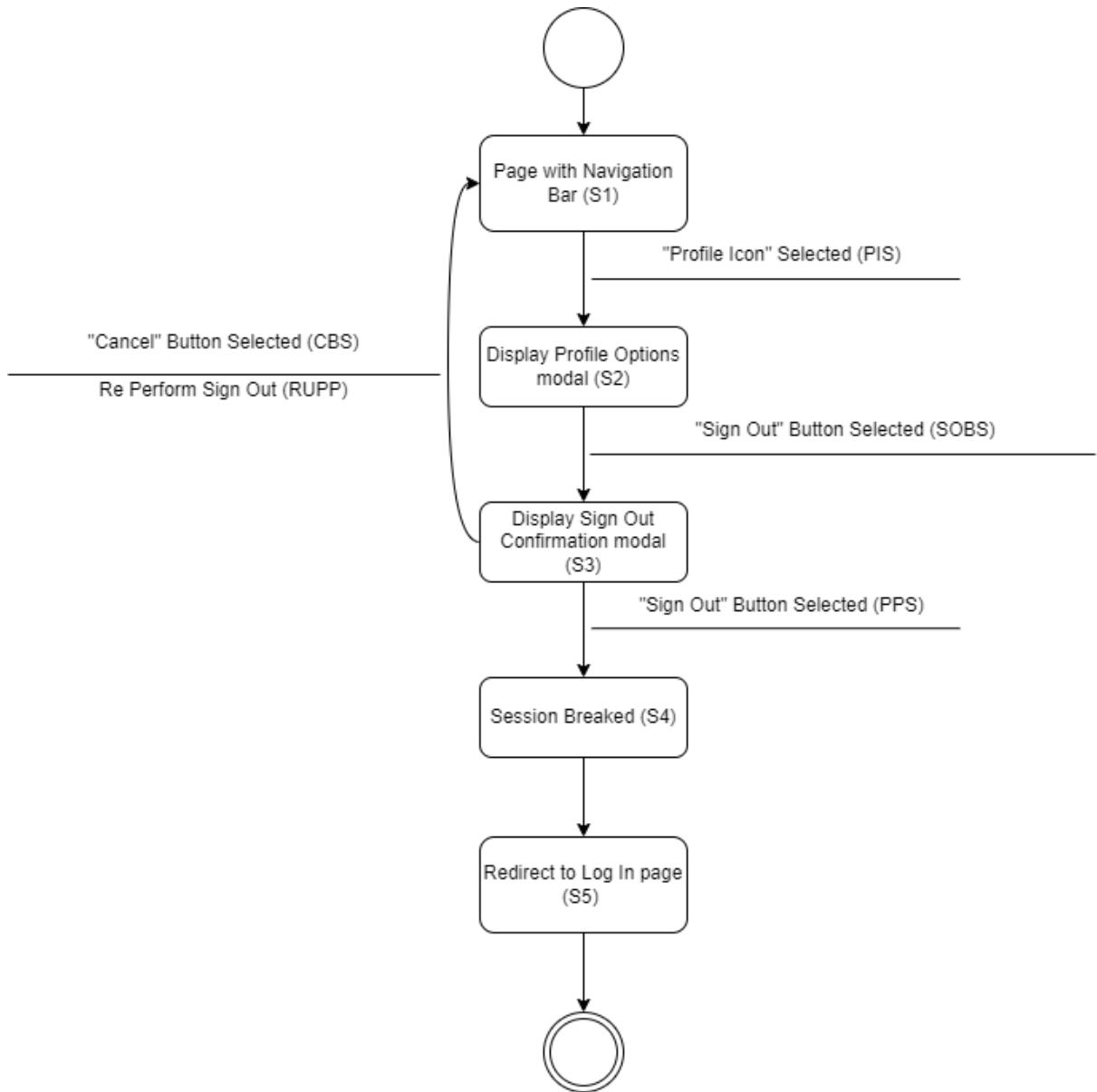


Figure 4. 37 State Chart Diagram – Sign Out

4.4.3 Hardware and Software Requirement

Hardware Requirement	
Hardware	Purpose
Laptop with internet access	This contributes to work documentation, system design, implementation, and development

Table 4. 21 Hardware Requirement

Software Requirement	
Software	Purpose
Microsoft Word	To document the system specifications and various UML diagrams associated with the system design
Drawio	To draw the UML diagrams
Uizard	To design user interface and contributes to feature identification
VSCode	To build up the environment for system development
Microsoft Edge (Web Browser)	To run the system and identify the real-time error of the system during the development process

Table 4. 22 Software Requirement

4.4.4 Product Features

ID	Feature	Description	Accessible Role
F001	Sign Up	To allow users to create an account to access to the system	User, Supplier
F002	Log In	To allow users or suppliers to login and access the system	User, Supplier
F003	Select Supplier	To allow users to select the preferred supplier	User
F004	Select Product	To allow users to select the preferred product	User
F005	Place Order	To allow users to place the order	User
F006	Check Order Status	To allow users to check order status	User, Supplier
F007	Check Invoice	To allow users or suppliers to check the invoice	User, Supplier
F008	Confirm Order	To allow suppliers to confirm the order	Supplier
F009	Edit Product	To allow suppliers to update product information or delete product	Supplier
F010	Add Product	To allow suppliers to add product to product list	Supplier
F011	Check Order Details	To allow users or suppliers to check order details before order has been confirmed	User, Supplier
F012	Add Company Description	To allow suppliers to update company description	Supplier
F013	Upload Profile Photo	To allow users or suppliers to upload profile photo	User, Supplier
F014	Search Supplier	To allow users to search the preferred supplier	User
F015	Search Order	To allow users or suppliers to search the order they want to check	User, Supplier
F016	Search Product	To allow suppliers to search the product they want to manage	Supplier
F017	Sign Out	To allow users or suppliers to sign out	User, Supplier

Table 4. 23 Product Features

4.4.5 Functional Requirements

Requirement ID	REQ_F001	Version	1.0
Description	System shall be able to allow users to create an account to access to the system.		
Author	Goo Han Cong		

Table 4. 24 REQ_F001

Requirement ID	REQ_F002	Version	1.0
Description	System shall be able to allow users, suppliers, or admins to login and access the system.		
Author	Goo Han Cong		

Table 4. 25 REQ_F002

Requirement ID	REQ_F003	Version	1.0
Description	System shall be able to allow users to select the preferred supplier.		
Author	Goo Han Cong		

Table 4. 26REQ_F003

Requirement ID	REQ_F004	Version	1.0
Description	System shall be able to allow users to select the preferred product.		
Author	Goo Han Cong		

Table 4. 27 REQ_F004

Requirement ID	REQ_F005	Version	1.0
Description	System shall be able to allow users to place the order.		
Author	Goo Han Cong		

Table 4. 28 REQ_F005

Requirement ID	REQ_F006	Version	1.0
Description	System shall be able to allow users to check order status.		
Author	Goo Han Cong		

Table 4. 29 REQ_F006

Requirement ID	REQ_F007	Version	1.0
Description	System shall be able to allow users or suppliers to check the invoice.		
Author	Goo Han Cong		

Table 4. 30 REQ_F007

Requirement ID	REQ_F008	Version	1.0
Description	System shall be able to suppliers to confirm the order.		
Author	Goo Han Cong		

Table 4. 31 REQ_F008

Requirement ID	REQ_F009	Version	1.0
Description	System shall be able to allow suppliers to update product information or delete product.		
Author	Goo Han Cong		

Table 4. 32 REQ_F009

Requirement ID	REQ_F010	Version	1.0
Description	System shall be able to allow suppliers to add product to product list.		
Author	Goo Han Cong		

Table 4. 33 REQ_F010

Requirement ID	REQ_F011	Version	1.0
Description	System shall be able to allow users or suppliers to check order details where the order has not been confirmed		
Author	Goo Han Cong		

Table 4. 34 REQ_F011

Requirement ID	REQ_F012	Version	1.0
Description	System shall be able to allow suppliers to update company description		
Author	Goo Han Cong		

Table 4. 35 REQ_F012

Requirement ID	REQ_F013	Version	1.0
Description	System shall be able to allow users or suppliers to upload profile photo		
Author	Goo Han Cong		

Table 4. 36 REQ_F013

Requirement ID	REQ_F014	Version	1.0
Description	System shall be able to allow users to search the preferred supplier		
Author	Goo Han Cong		

Table 4. 37 REQ_F014

Requirement ID	REQ_015	Version	1.0
Description	System shall be able to allow users or suppliers to search the order they want to check		
Author	Goo Han Cong		

Table 4. 38 REQ_F015

Requirement ID	REQ_F016	Version	1.0
Description	System shall be able to allow suppliers to search the product they want to manage		
Author	Goo Han Cong		

Table 4. 39 REQ_F016

Requirement ID	REQ_F017	Version	1.0
Description	System shall be able to allow users or suppliers to sign out		
Author	Goo Han Cong		

Table 4. 40 REQ_F017

4.4.6 Non-Functional Requirements

4.4.6.1 Usability System Requirements

ID	Ver	Description	Priority
001-001	1.0	Understandability	
001-001-001	1.0	The system shall use plain language, without gaps of confusion, in error messages, buttons, and text field labels.	7
001-002	1.0	Operability	
001-002-001	1.0	The system shall be easy to use.	7
001-002-002	2.0	The system shall have fast response when executing.	8
001-003	1.0	Memorability	
001-003-001	1.0	The system shall have consistent interface that allow user to remember when operating the tasks.	8

Table 4. 41 Non-Functional Requirements - Usability

4.4.6.2 Reliability System Requirements

ID	Ver	Description	Priority
002-001	1.0	Reliability	
002-001-001	1.0	The system shall be reliable with no crashing and error.	9

Table 4. 42 Non-Functional Requirements – Reliability

4.4.6.3 Functionality System Requirements

ID	Ver	Description	Priority
003-001	1.0	Accuracy	
003-001-001	1.0	The system shall ensure that information including user, product, are all recorded accurately.	9
003-002	1.0	Suitability	
003-002-001	1.0	The system shall be in English.	7
003-003	1.0	Security	
003-003-001	1.0	The system shall encrypt users' account passwords using a hash algorithm before storing their data onto the database.	8
003-003-002	1.0	The system shall implement authentication to prevent unauthorized access, including protection against route manipulation	9

Table 4. 43 Non-Functional Requirements – Functionality

4.4.6.4 Maintainability System Requirements

ID	Ver	Description	Priority
004-001	1.0	Analysability	
004-001-001	1.0	The system's development and maintenance shall ensure that all requirements are linked to all tests.	9
004-002	1.0	Compliance	
004-002-001	1.0	The system shall meet the users' expected performance.	7

Table 4. 44 Non-Functional Requirements – Maintainability

4.4.6.5 Efficiency System Requirements

ID	Ver	Description	Priority
005-001	1.0	Resource Utilization	
005-001-001	1.0	The system shall be able to support up to 5,000 users.	7

Table 4. 45 Non-Functional Requirements – Efficiency

4.4.6.6 Portability System Requirements

ID	Ver	Description	Priority
006-001	1.0	Accessibility	
006-001-001	1.0	The system shall be compatible with major operating system such as Windows, macOS and Linux.	7
006-001-002	1.0	The system shall be accessible from various screen size of devices, such as laptops and tablet.	7

Table 4. 46 Non-Functional Requirements – Portability

Chapter 5 Design

In this chapter, the technical domain and the design structure of the system is explored via various diagram and tools. UML is also known as Unified Modeling Language which provides the ability to visualize the system structure, information flow, interaction among system components and so on. These diagrams include class diagram, activity diagrams, sequence diagrams, and state chart diagrams. Additionally, Entity-Relationship Diagrams (ERD) and relational schemas are utilized to model the database structure and relationships.

They all contribute to the understanding off comprehensive functionality of the system from a higher level. Design principle, system interface prototypes are also the crucial elements discussed in this chapter.

In summary, this chapter put the priority on understanding the system through UML diagram and the other visualization techniques.

5.1 Technical Drawings

5.1.1 Class Diagram

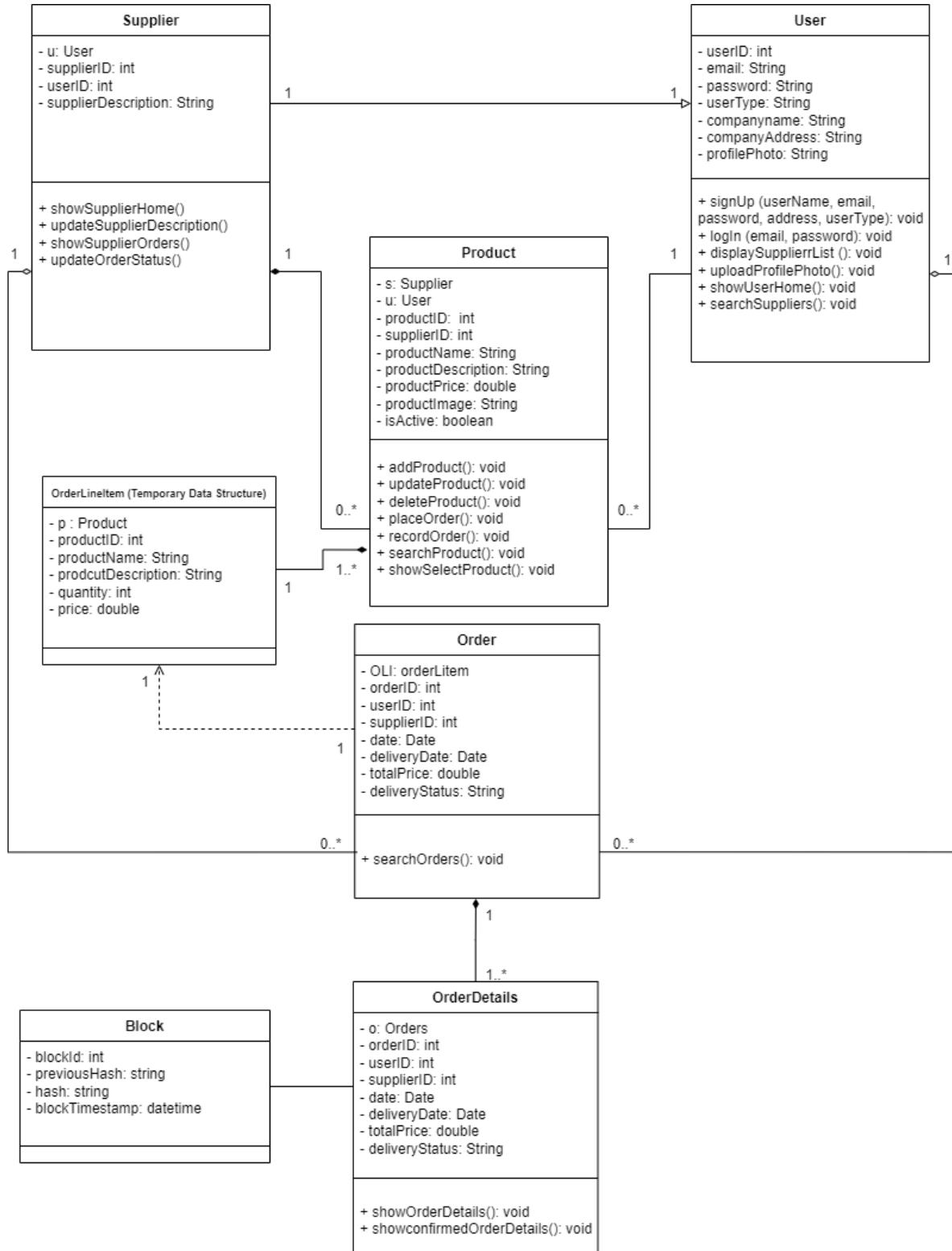


Figure 5. 1 Class Diagram

5.1.2 Entity Relationship Diagram

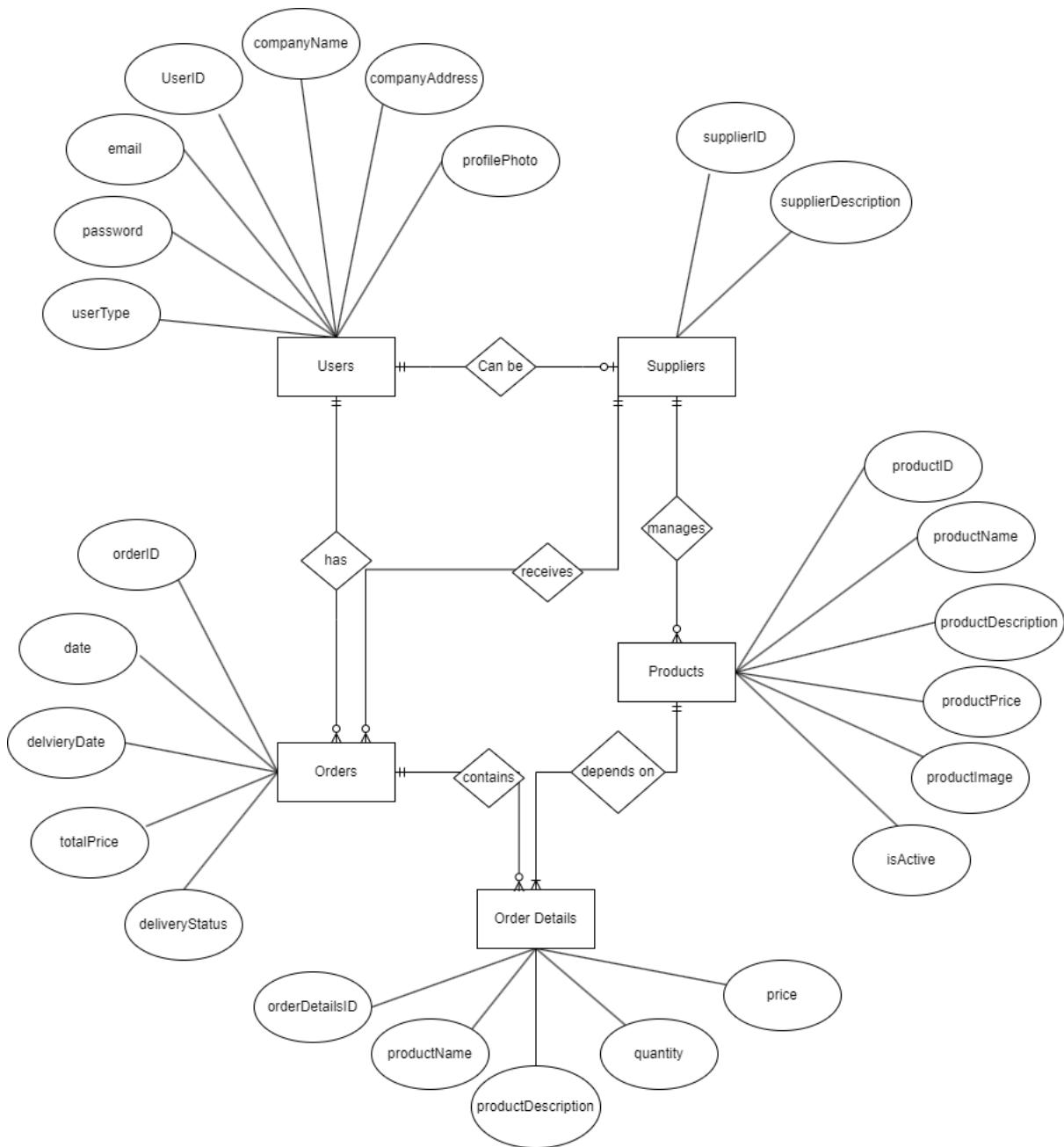


Figure 5. 2 Entity Relationship Diagram

5.1.3 Relational Schema

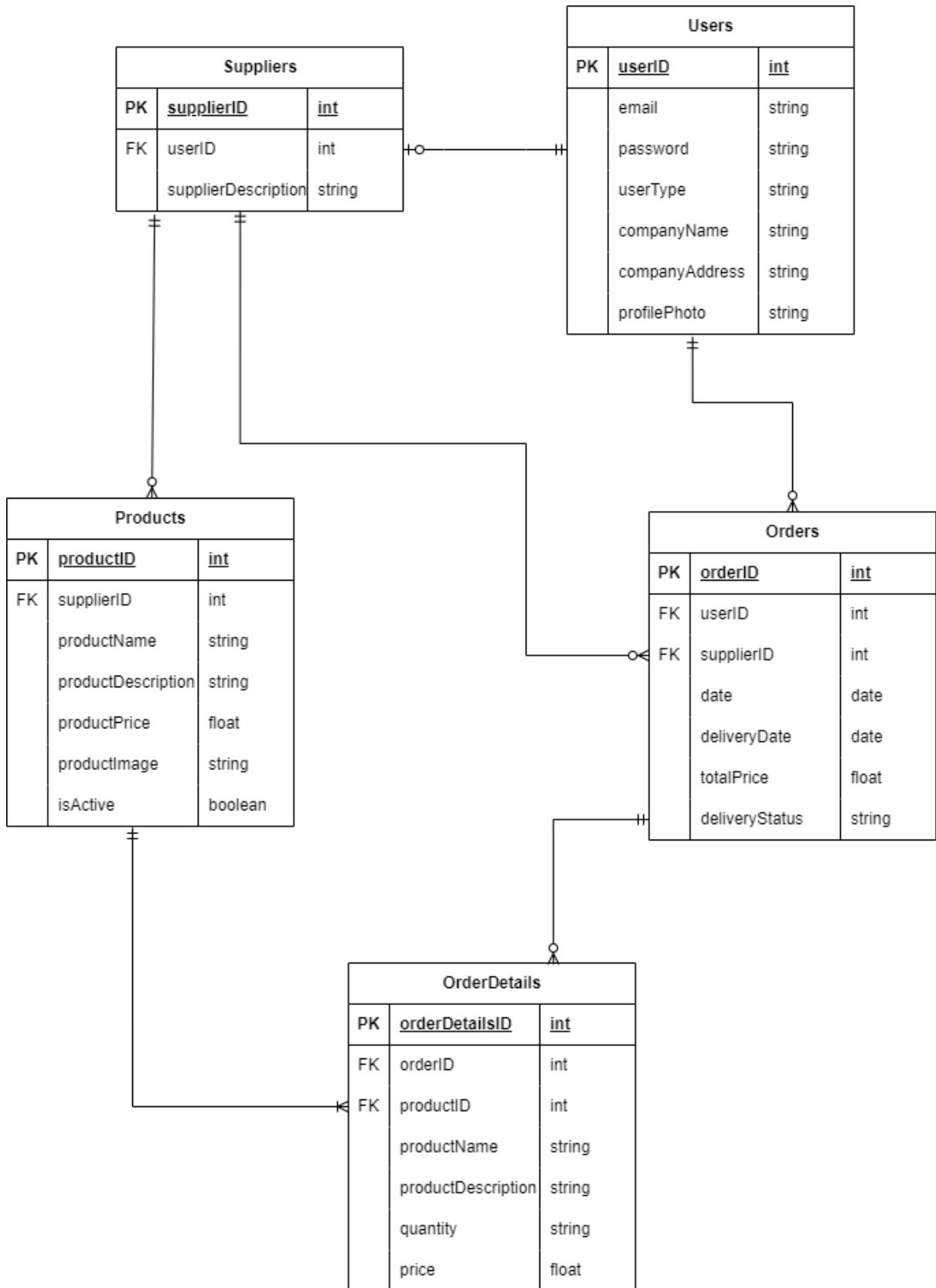


Figure 5. 3 Relational Schema

5.1.4 Sequence Diagram

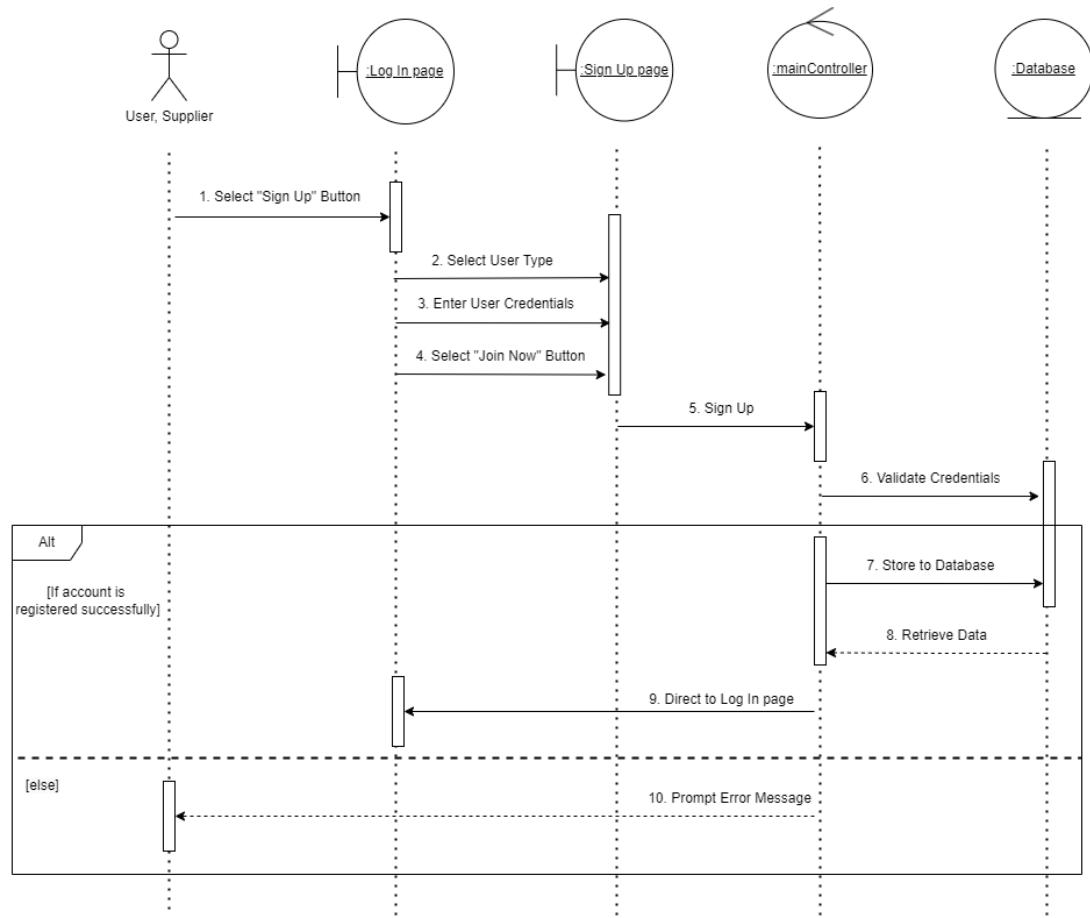


Figure 5. 4 Sequence Diagram – Sign Up

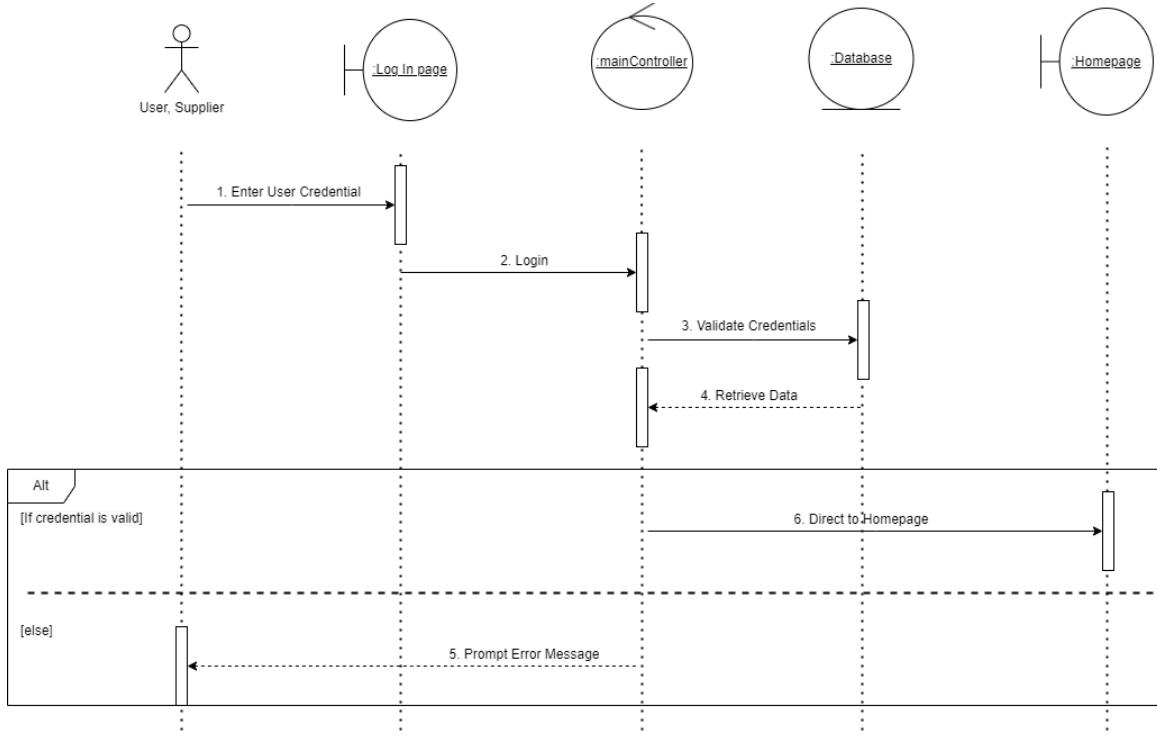


Figure 5. 5 Sequence Diagram – Log In

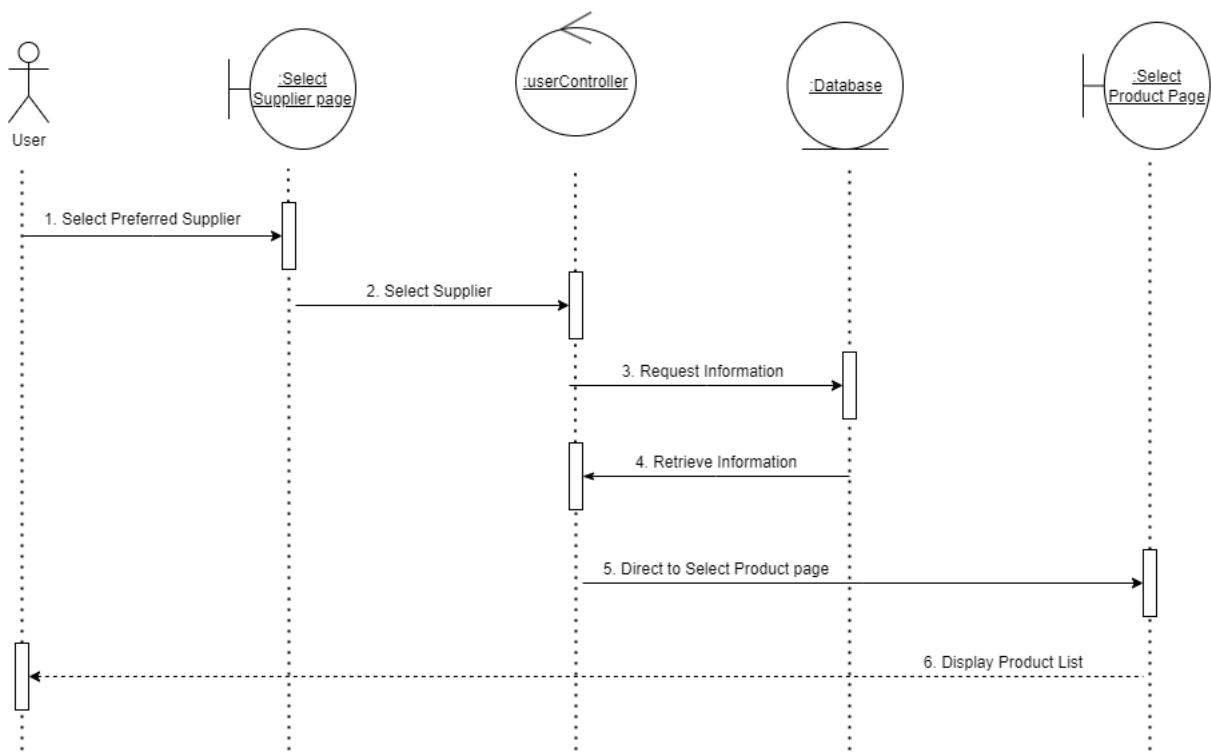


Figure 5. 6 Sequence Diagram – Select Supplier

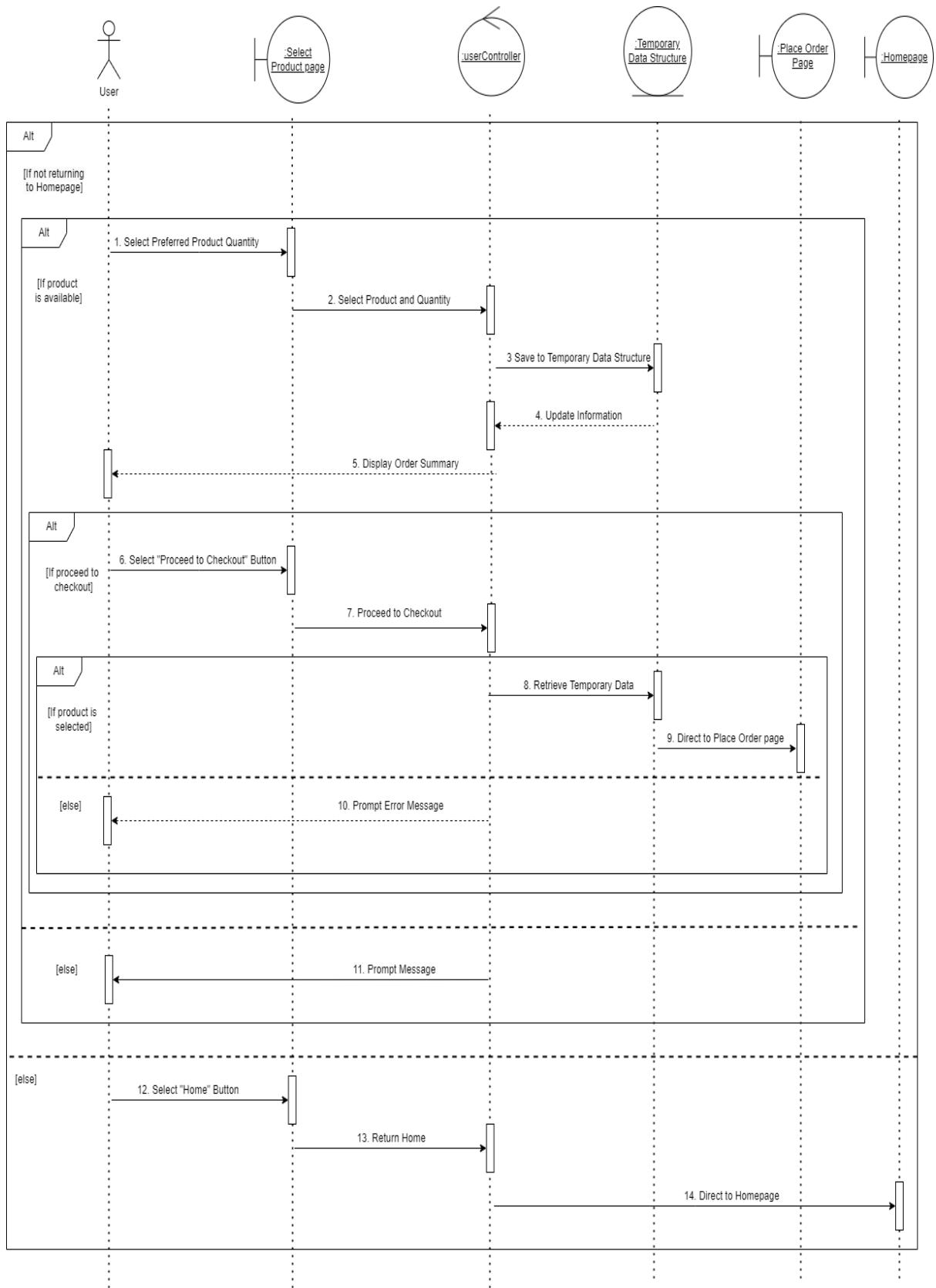


Figure 5. 7 Sequence Diagram – Select Product

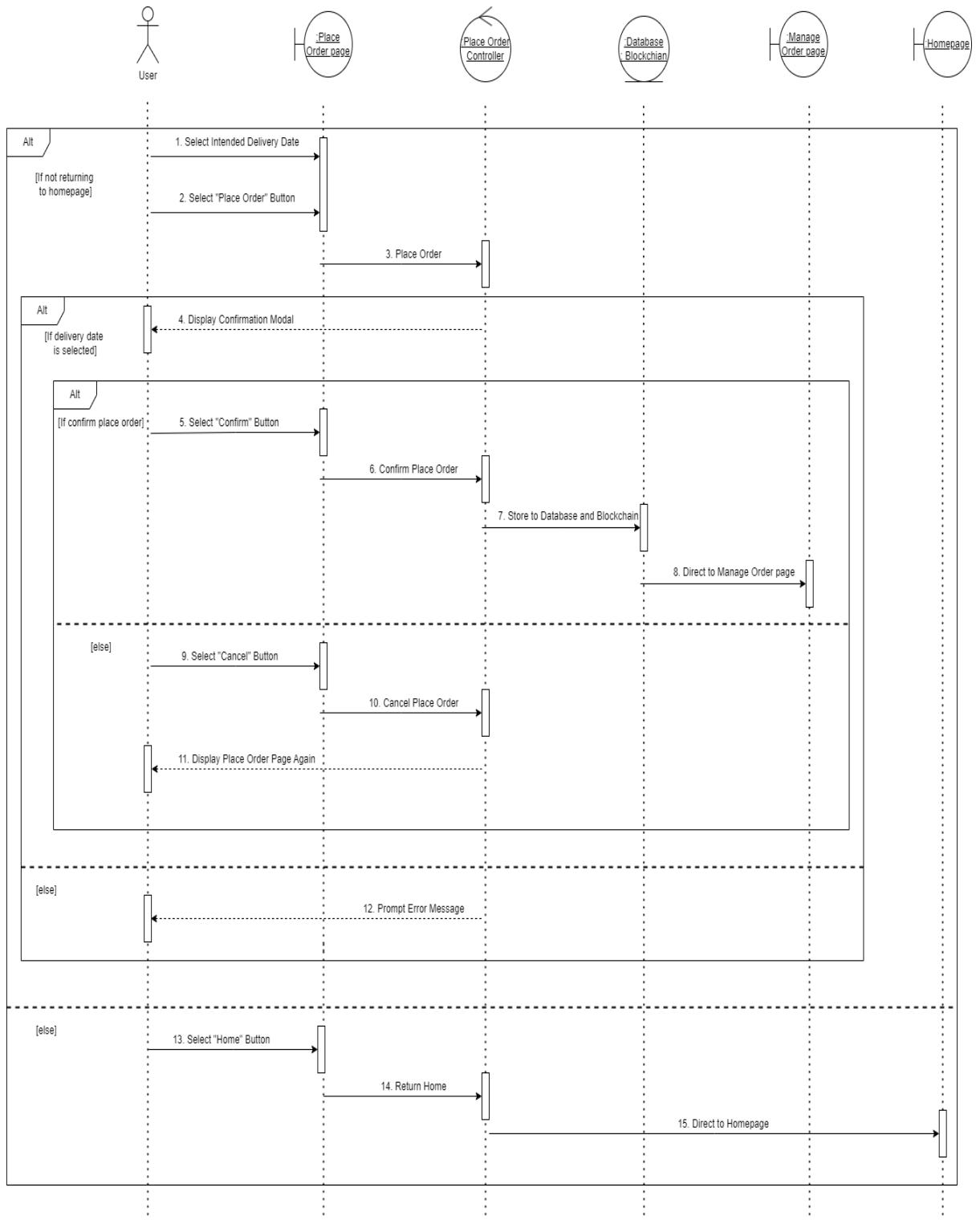


Figure 5. 8 Sequence Diagram – Place Order

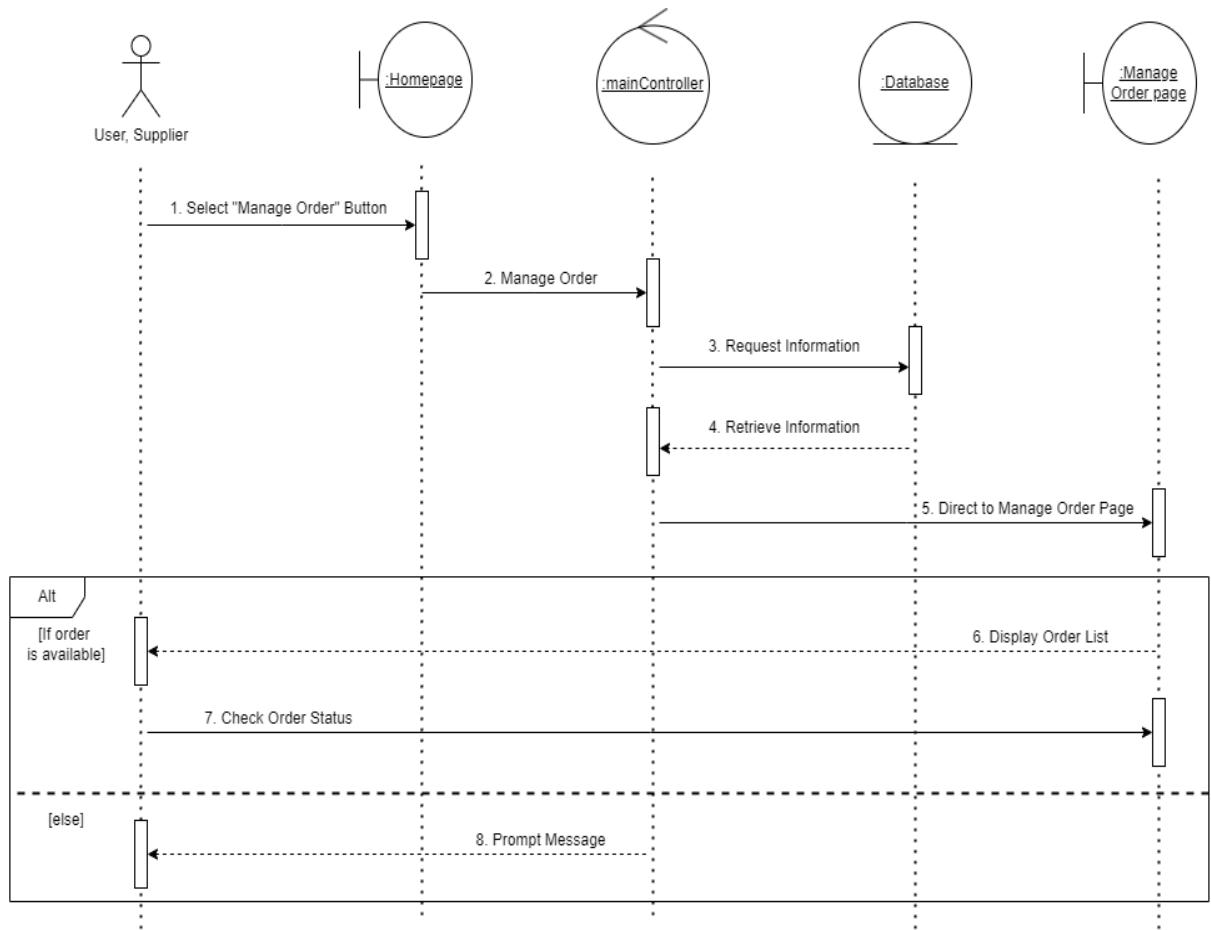


Figure 5. 9 Sequence Diagram – Check Order Status

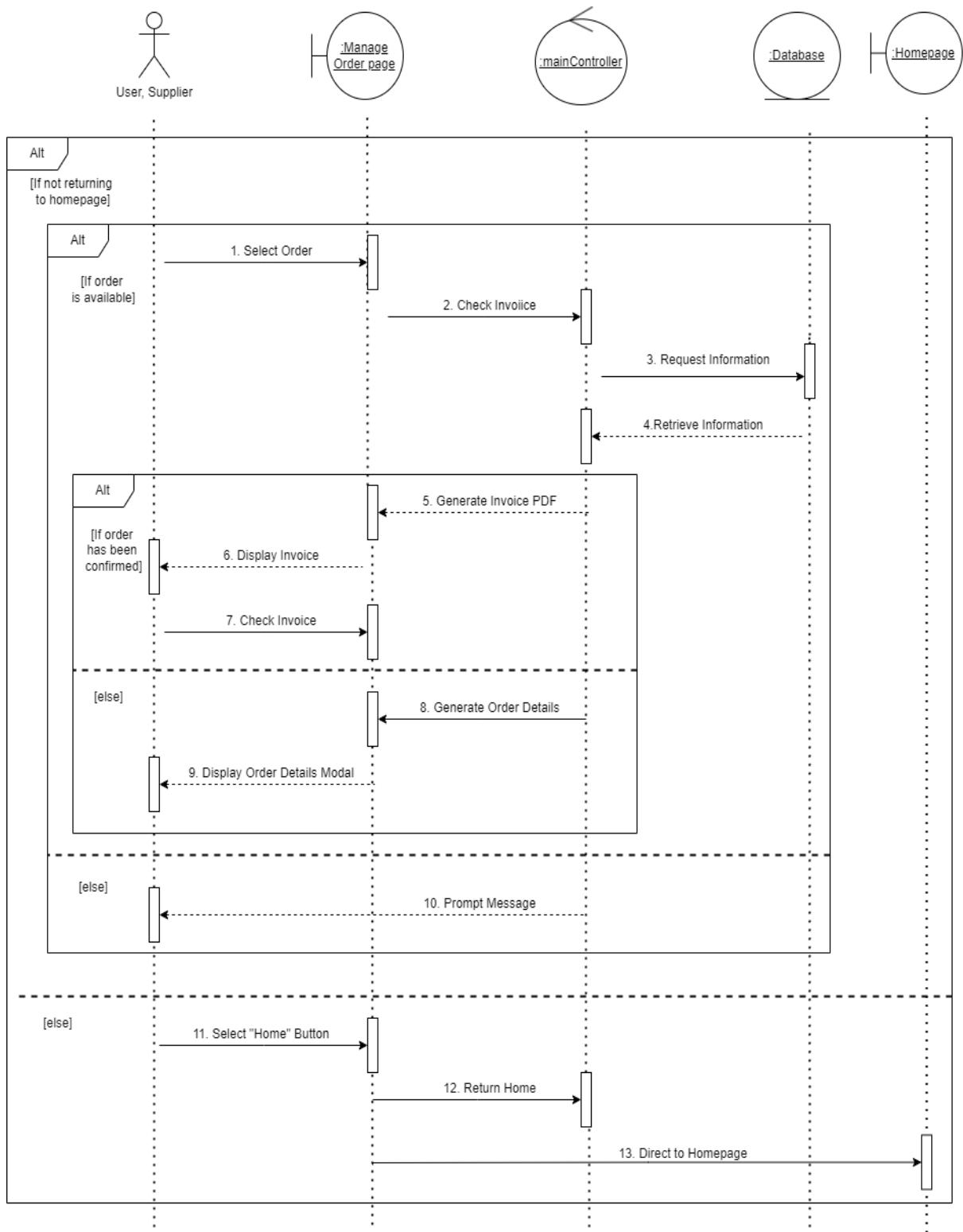


Figure 5. 10 Sequence Diagram – Check Invoice

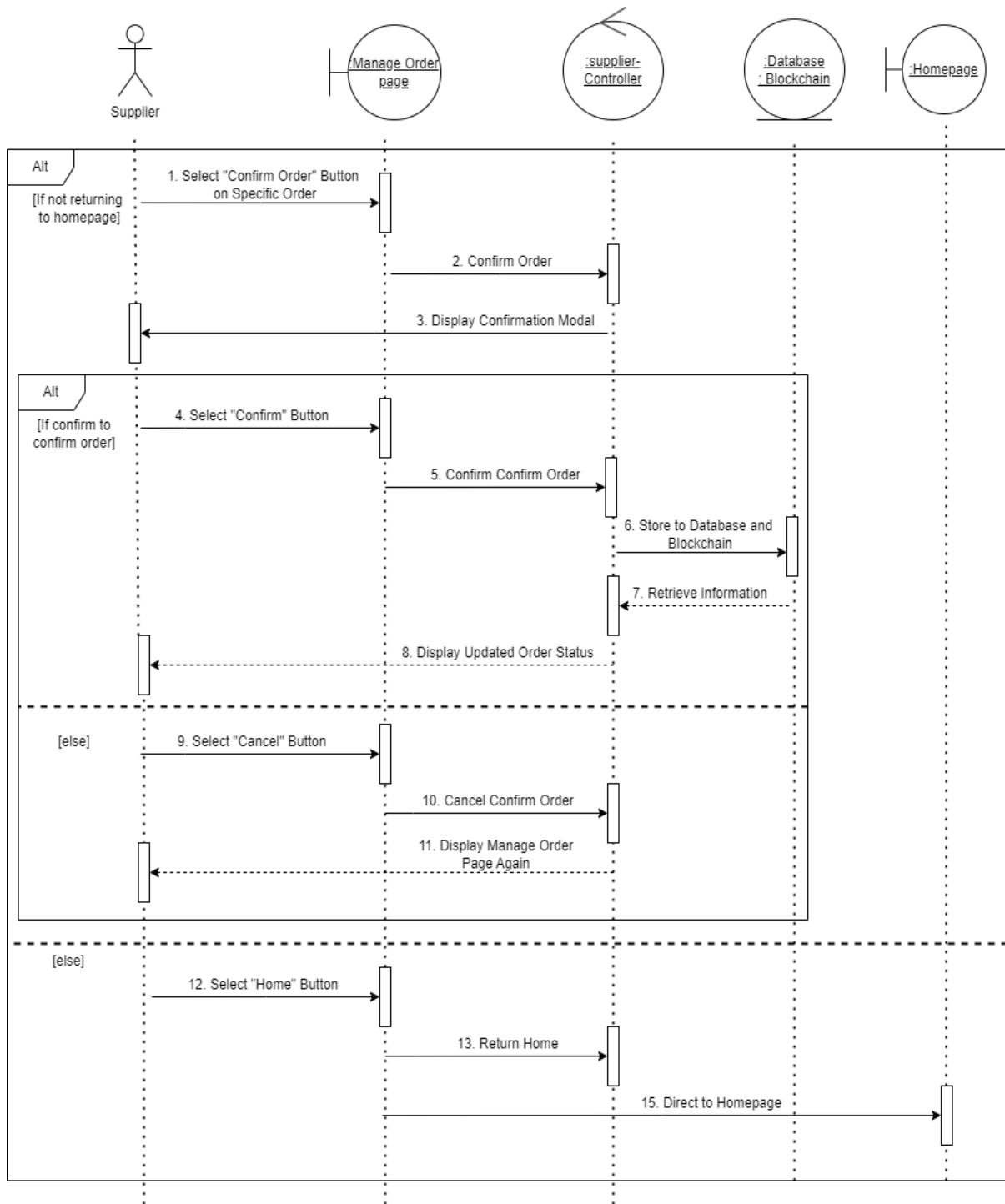


Figure 5. 11 Sequence Diagram – Confirm Order

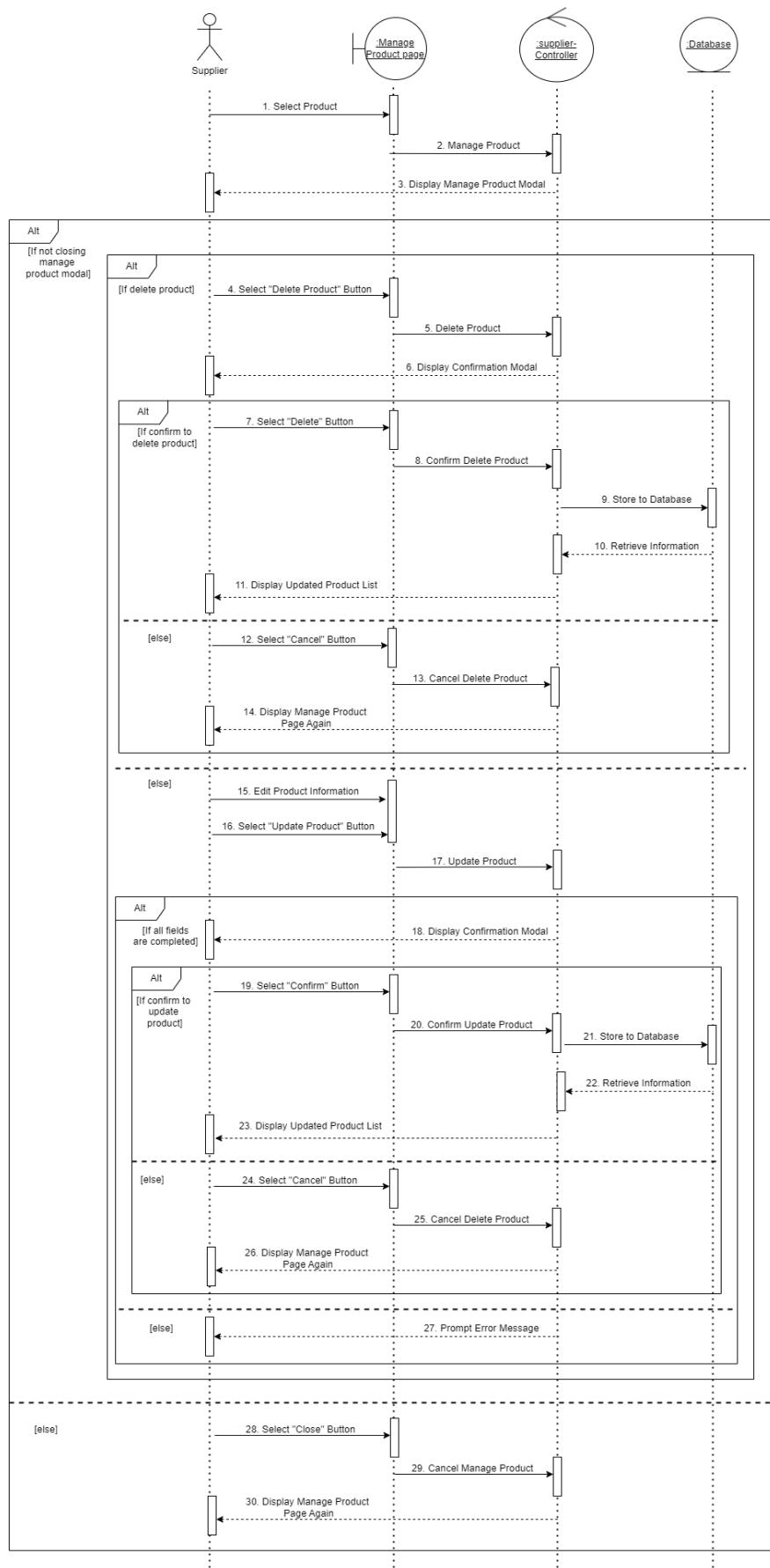


Figure 5. 12 Sequence Diagram – Edit Product

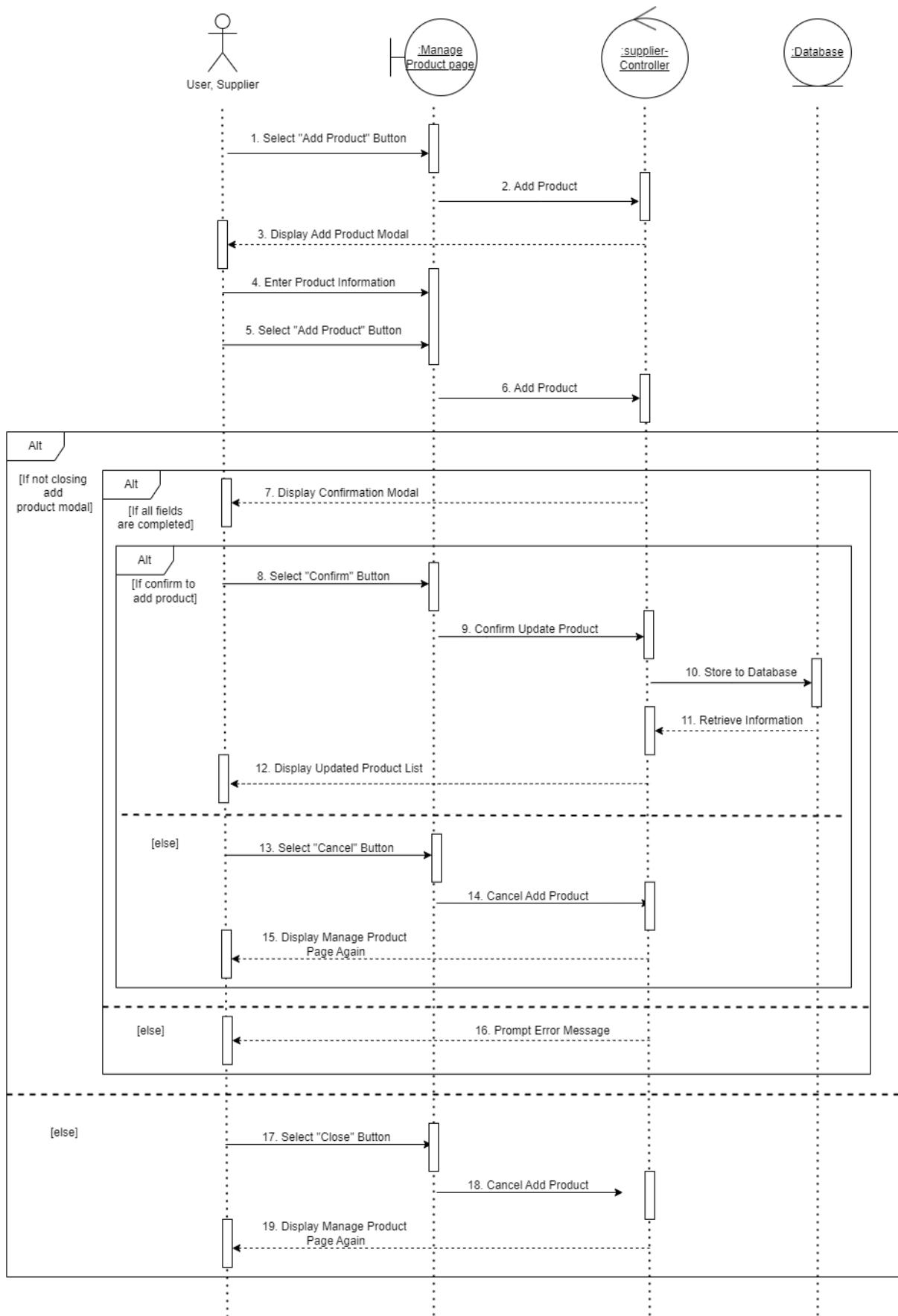


Figure 5.13 Sequence Diagram – Add Product

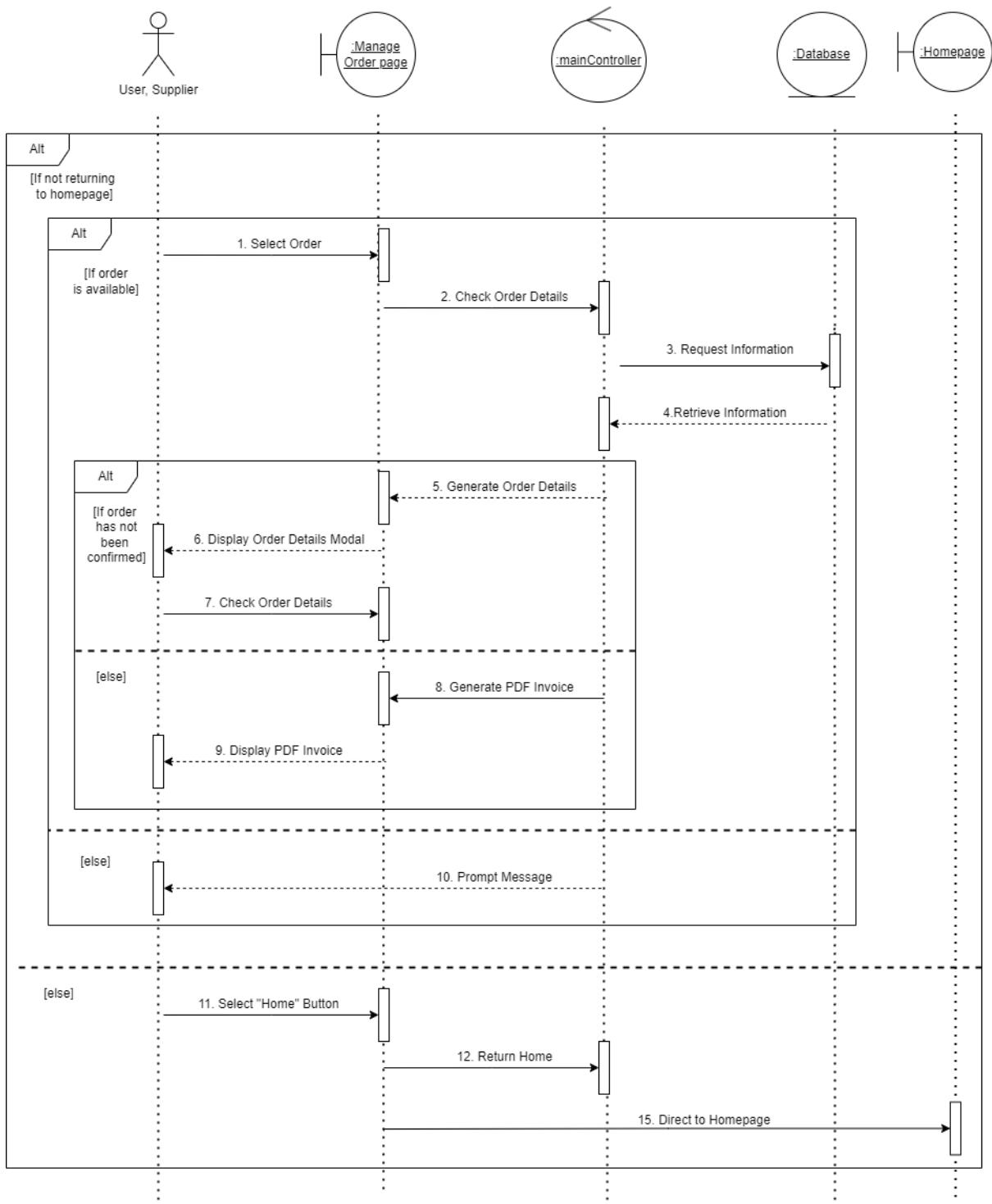


Figure 5. 14 Sequence Diagram – Check Order Details

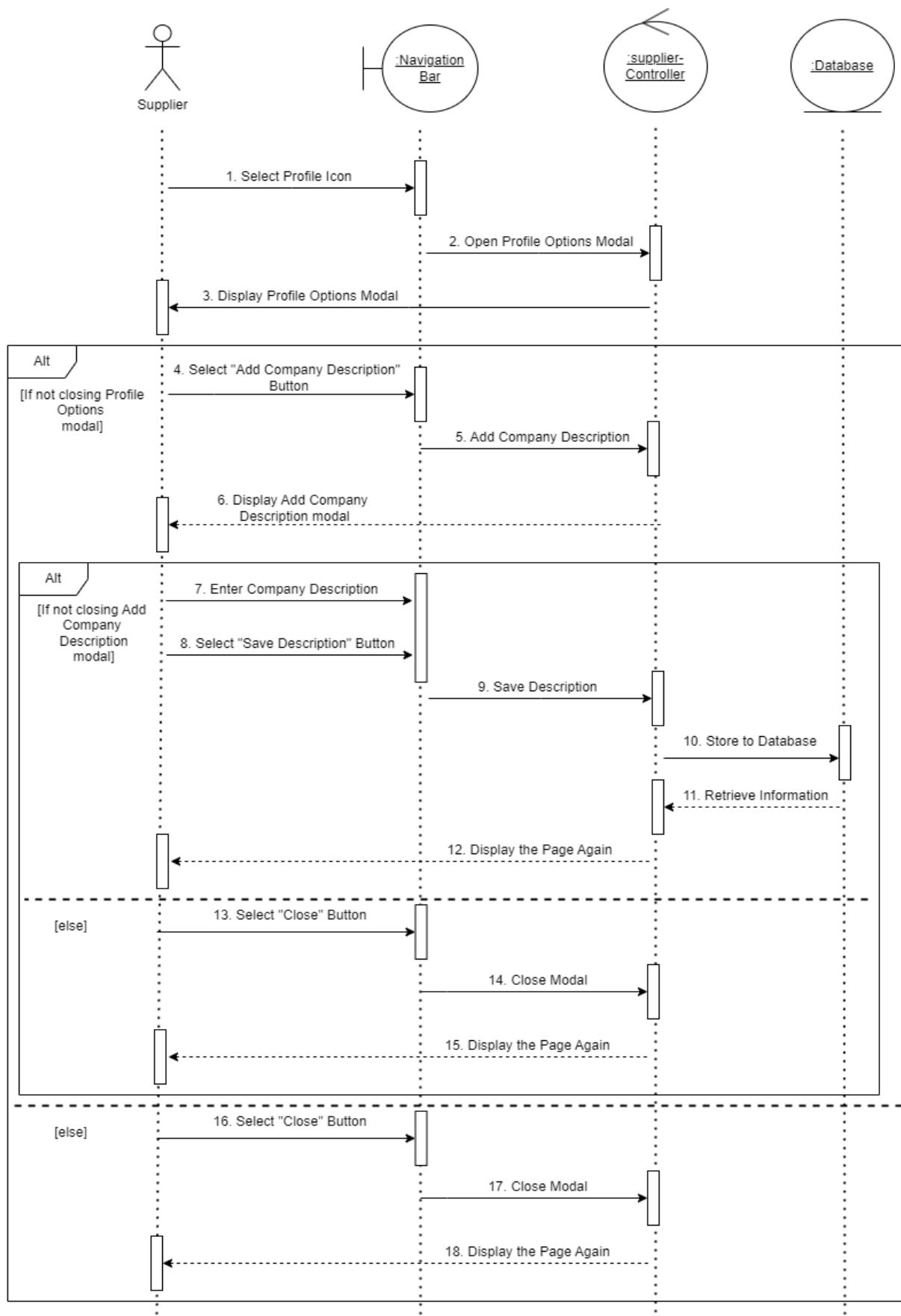


Figure 5. 15 Sequence Diagram – Add Company Description

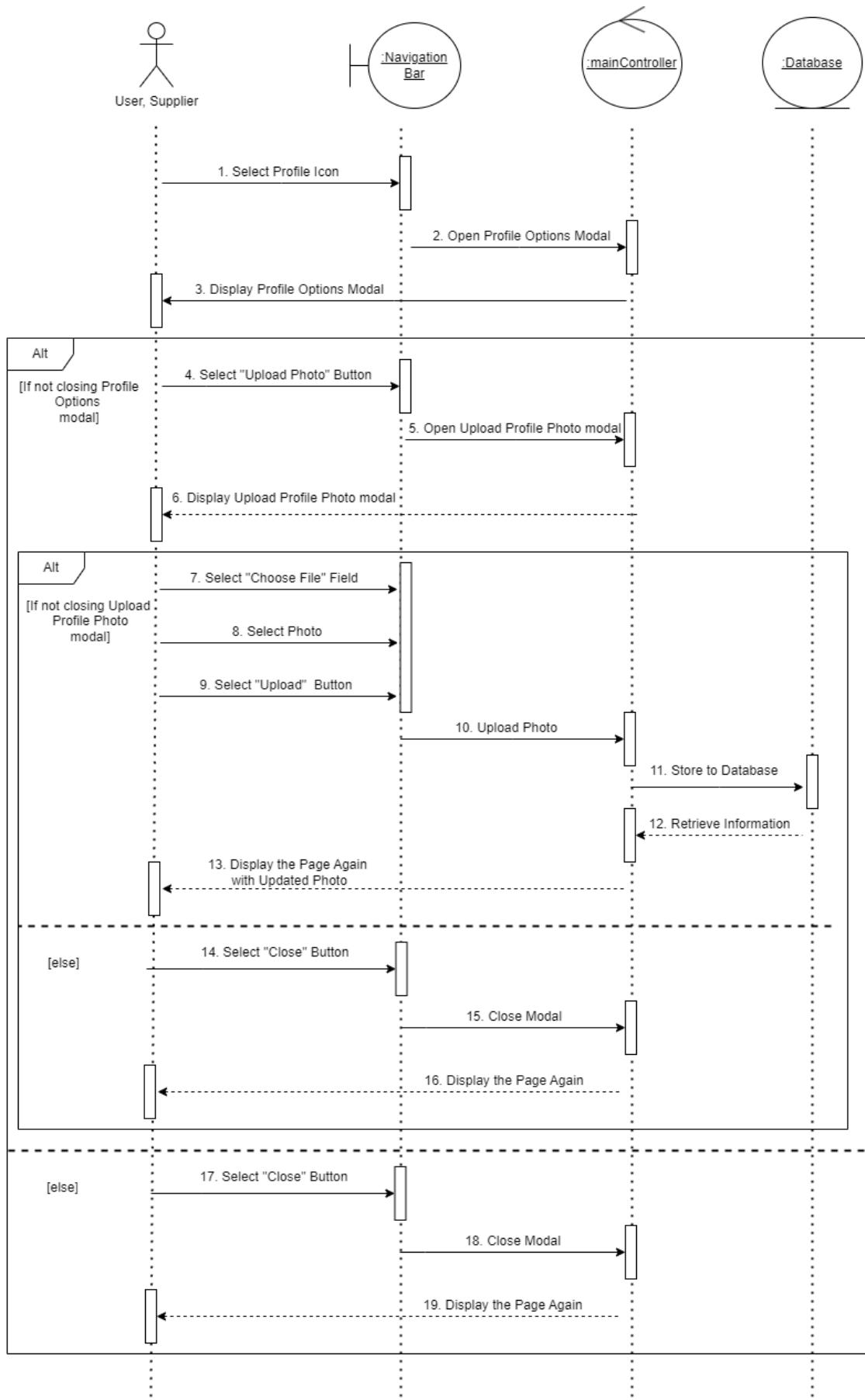


Figure 5. 16 Sequence Diagram – Upload Profile Photo

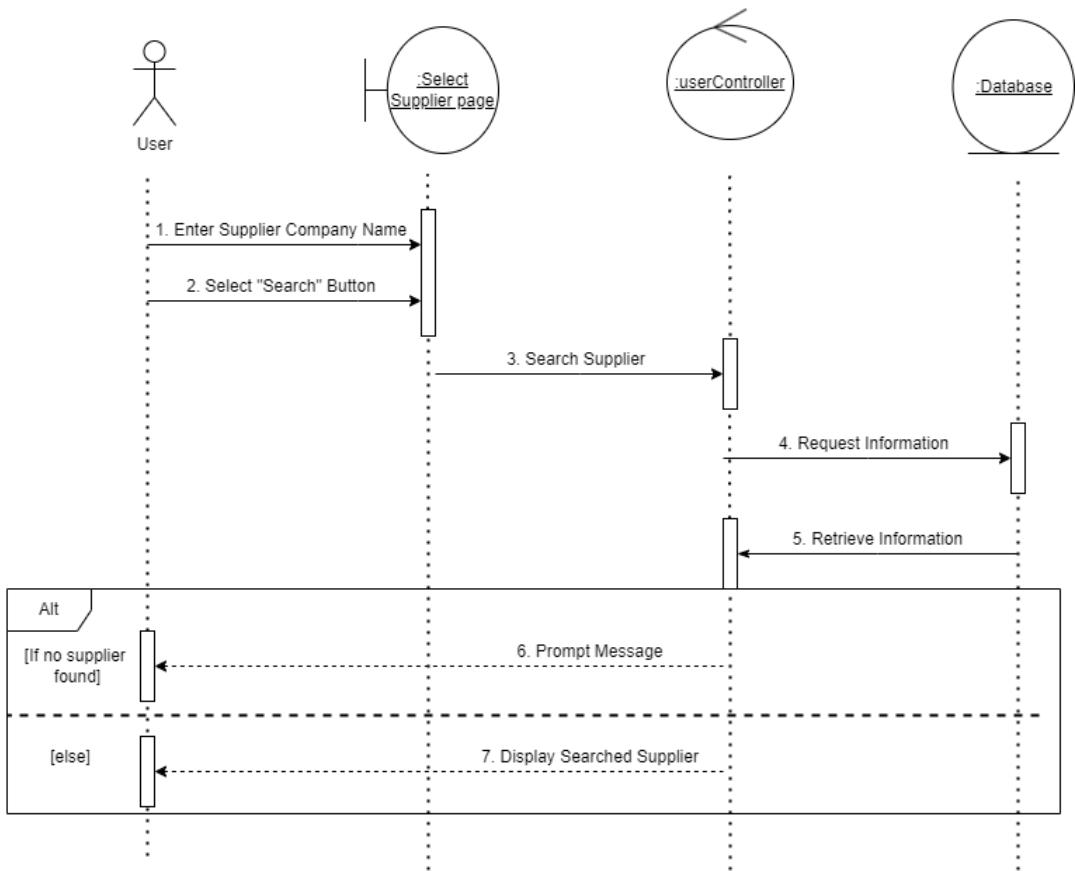


Figure 5. 17 Sequence Diagram – Search Supplier

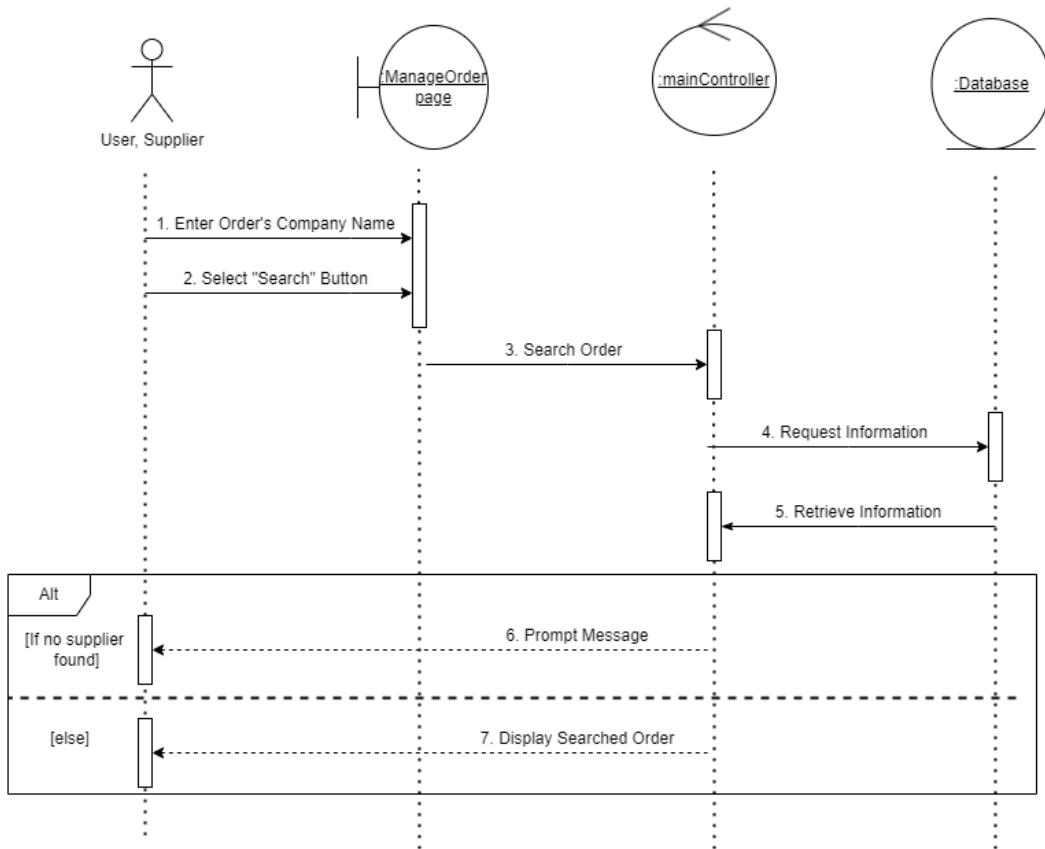


Figure 5. 18 Sequence Diagram – Search Order

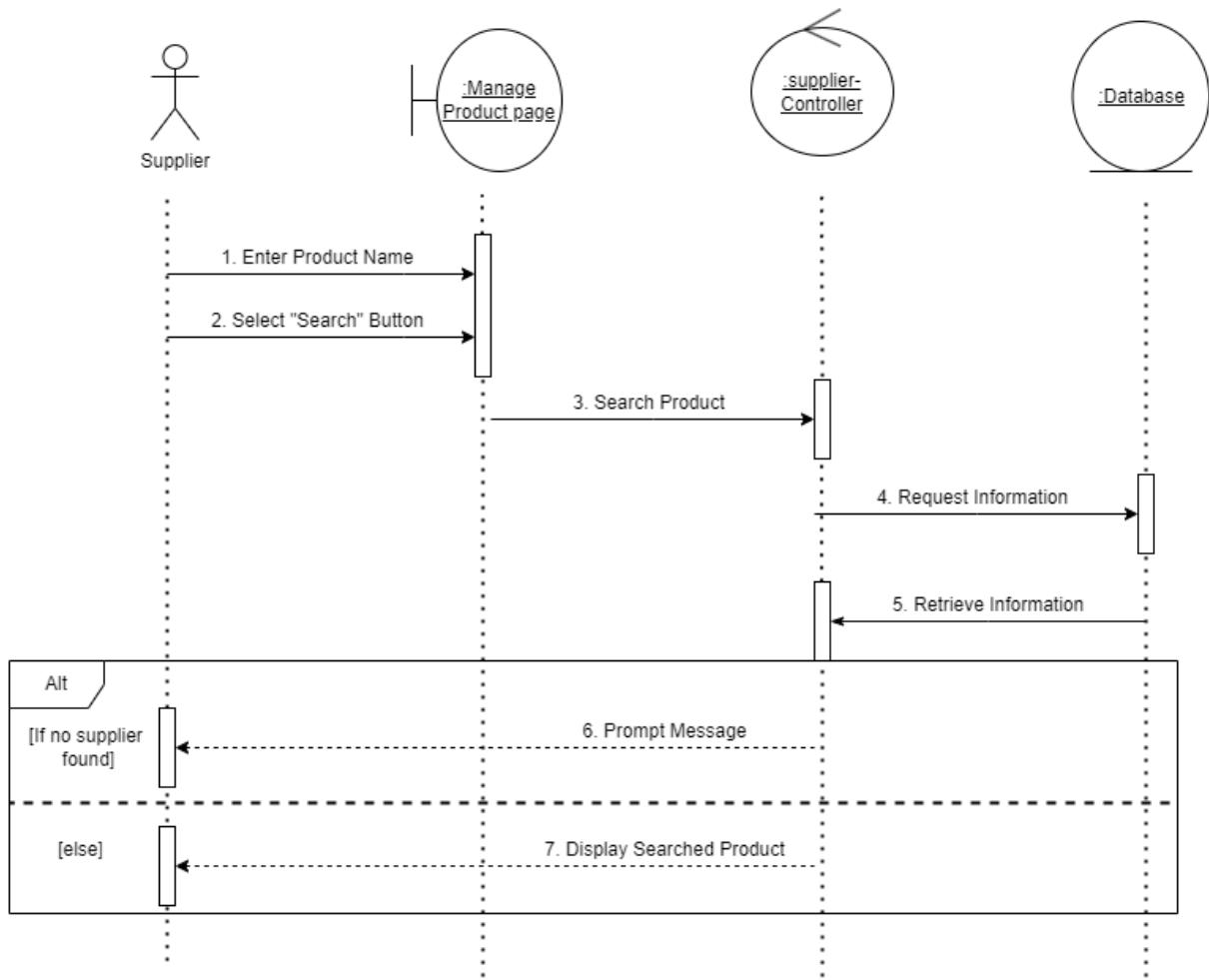


Figure 5. 19 Sequence Diagram – Search Product

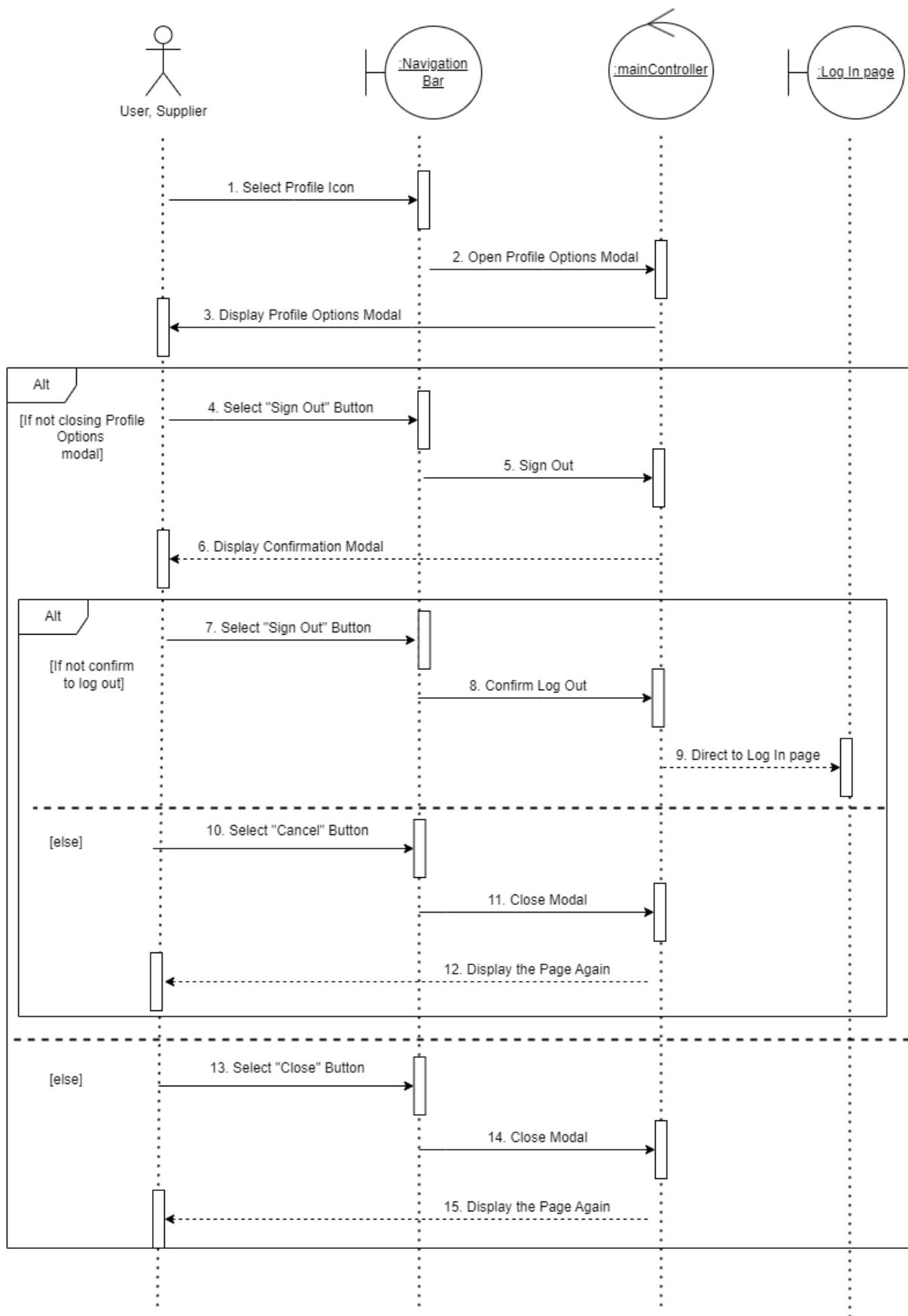


Figure 5. 20 Sequence Diagram – Sign Out

5.1.5 Activity Diagram

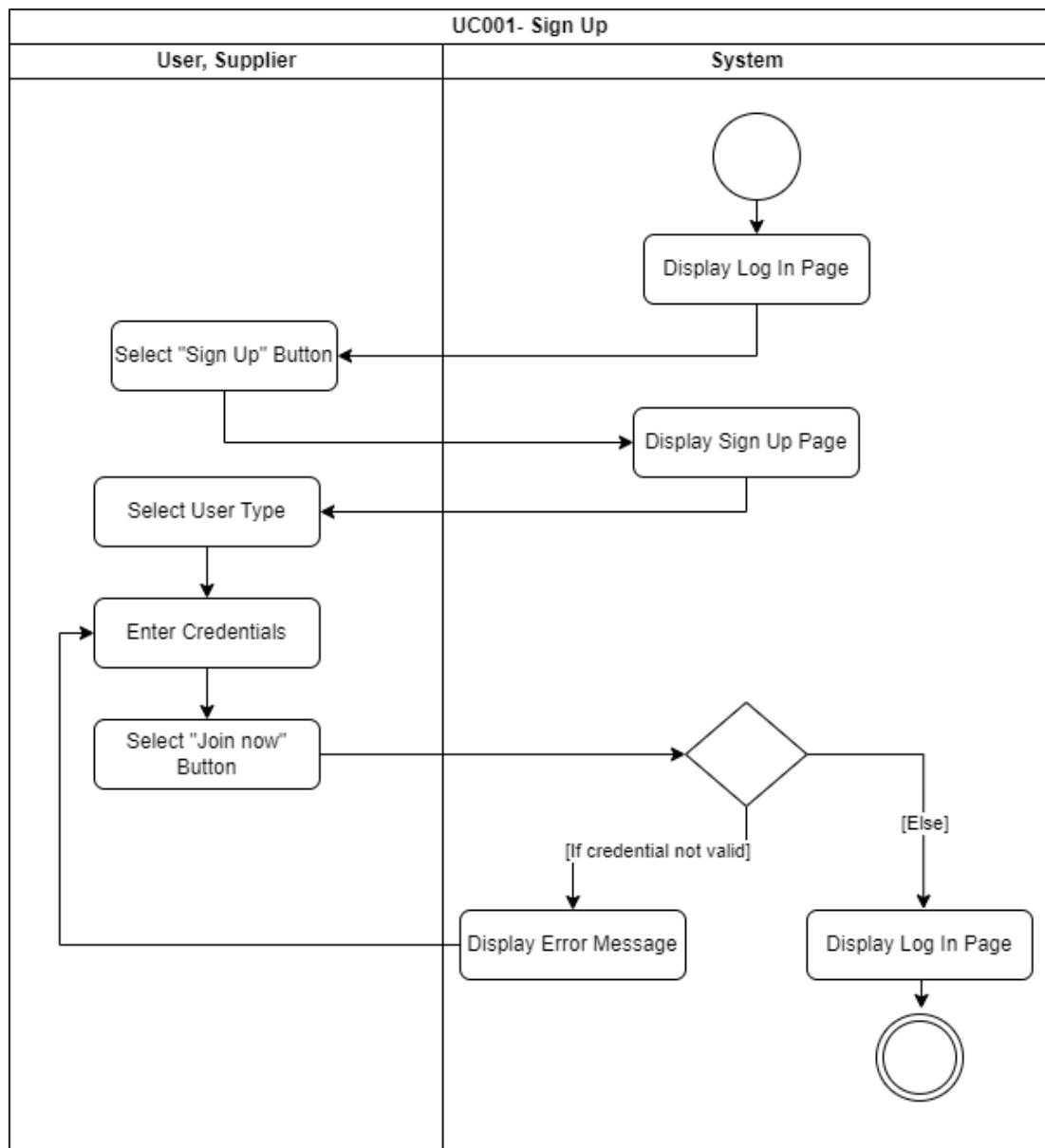


Figure 5. 21 Activity Diagram – Sign Up

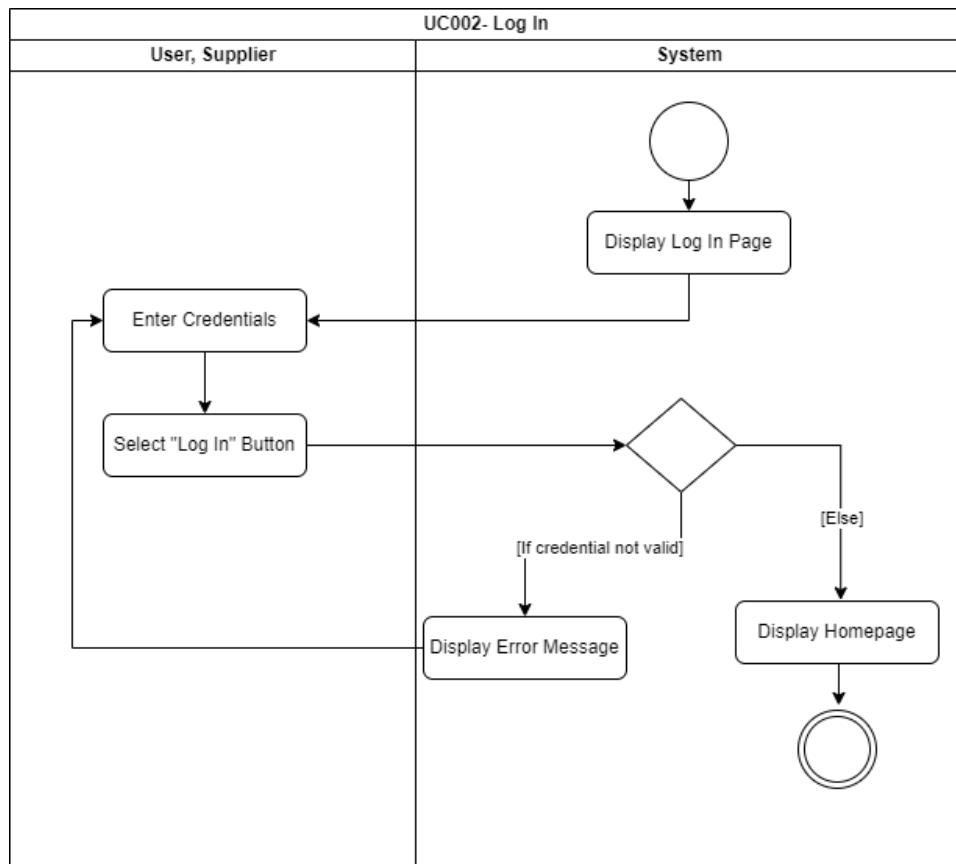


Figure 5. 22 Activity Diagram – Log In

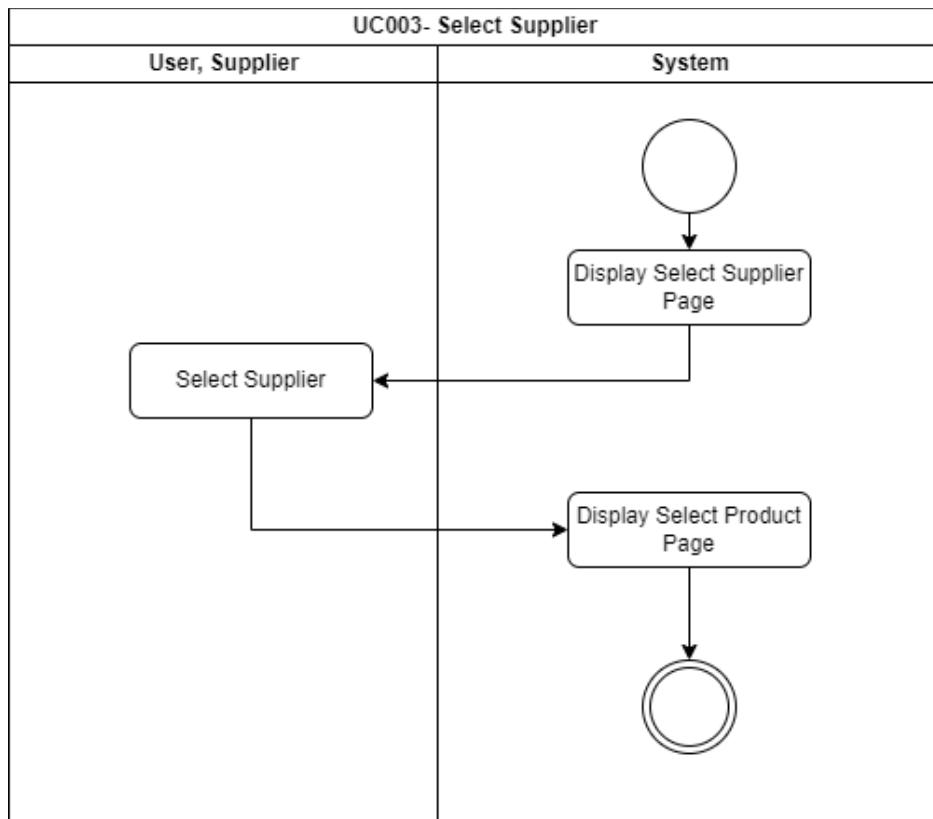


Figure 5. 23 Activity Diagram – Select Supplier

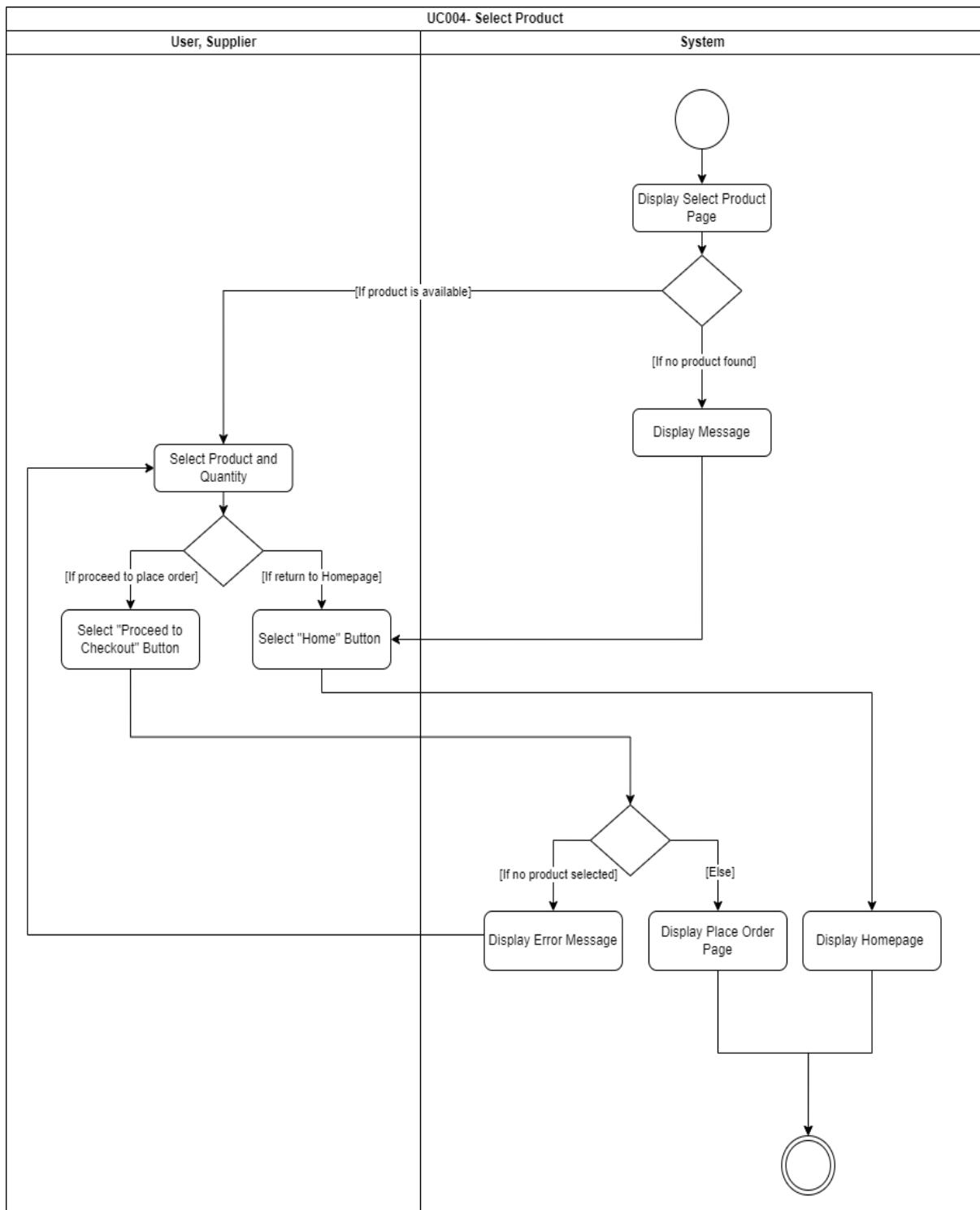


Figure 5. 24 Activity Diagram – Select Product

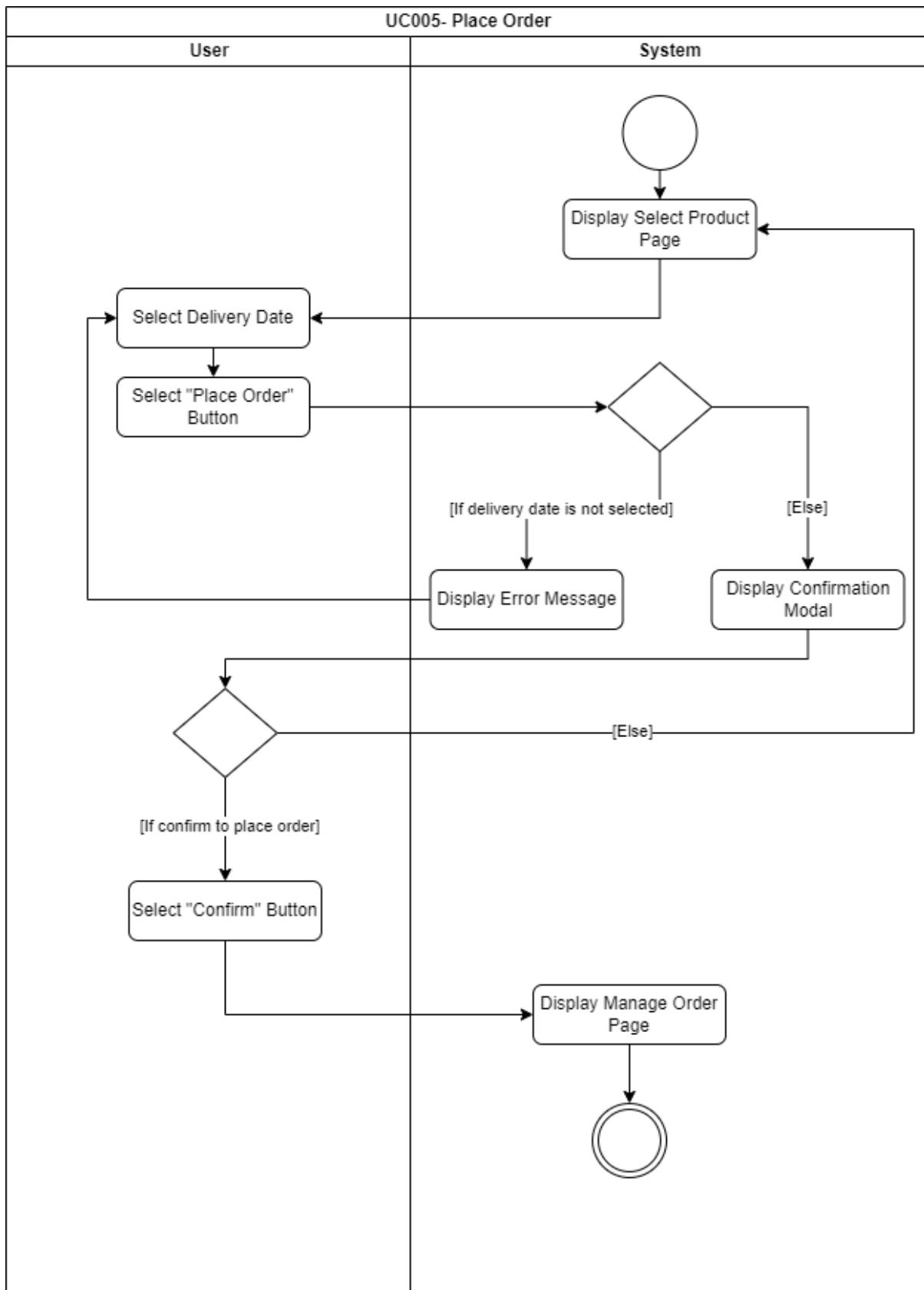


Figure 5. 25 Activity Diagram – Place Order

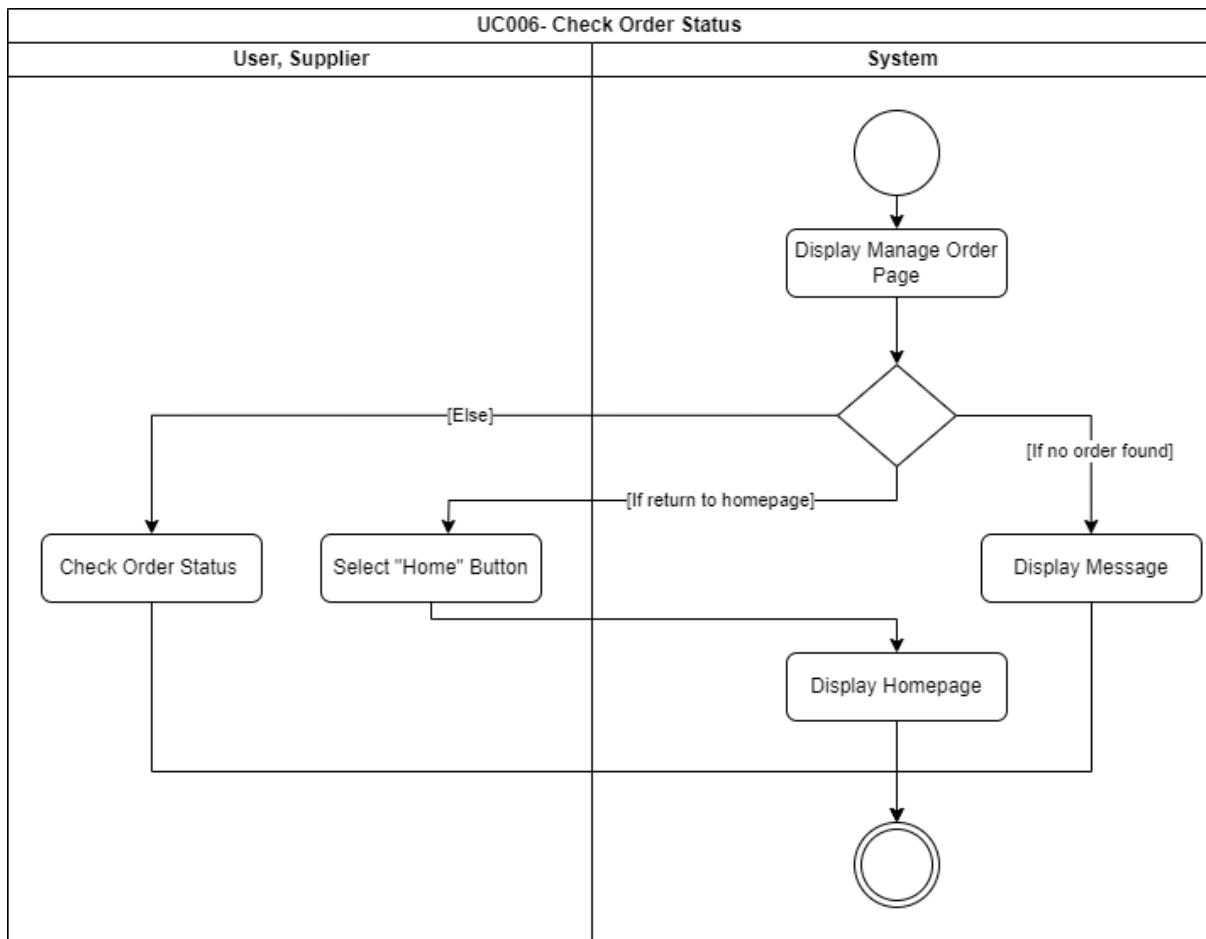


Figure 5. 26 Activity Diagram – Check Order Status

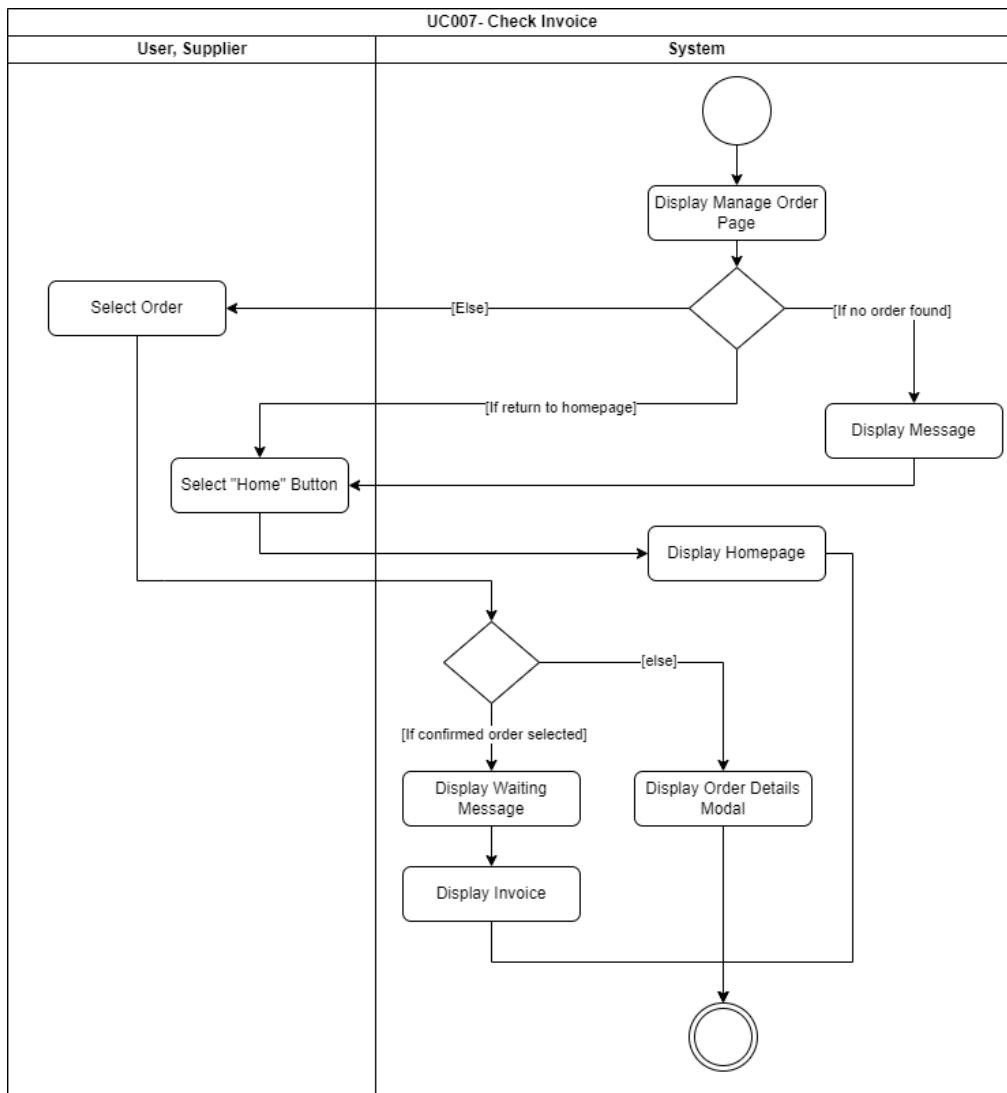


Figure 5. 27 Activity Diagram – Check Invoiice

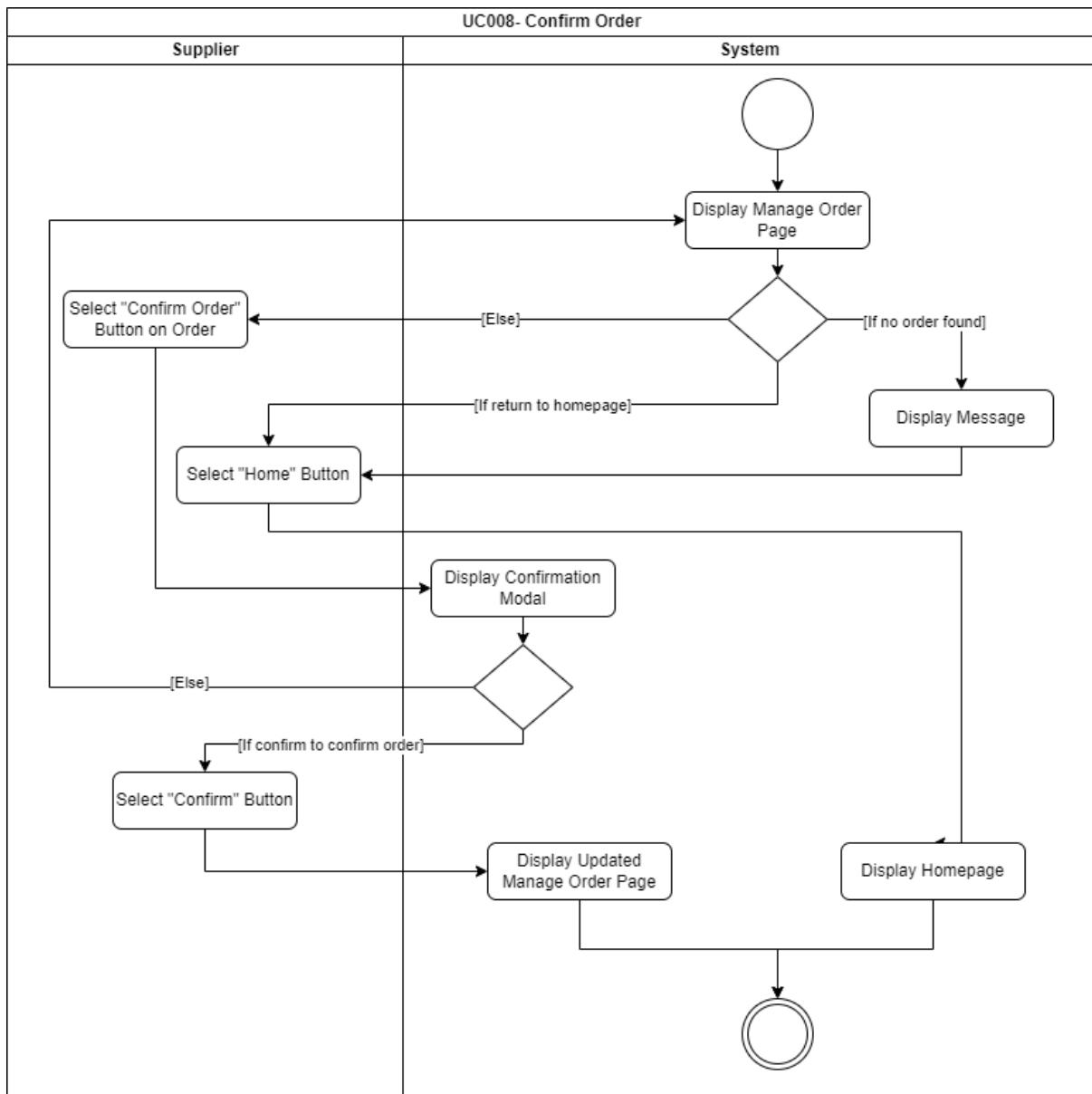


Figure 5. 28 Activity Diagram – Confirm Order

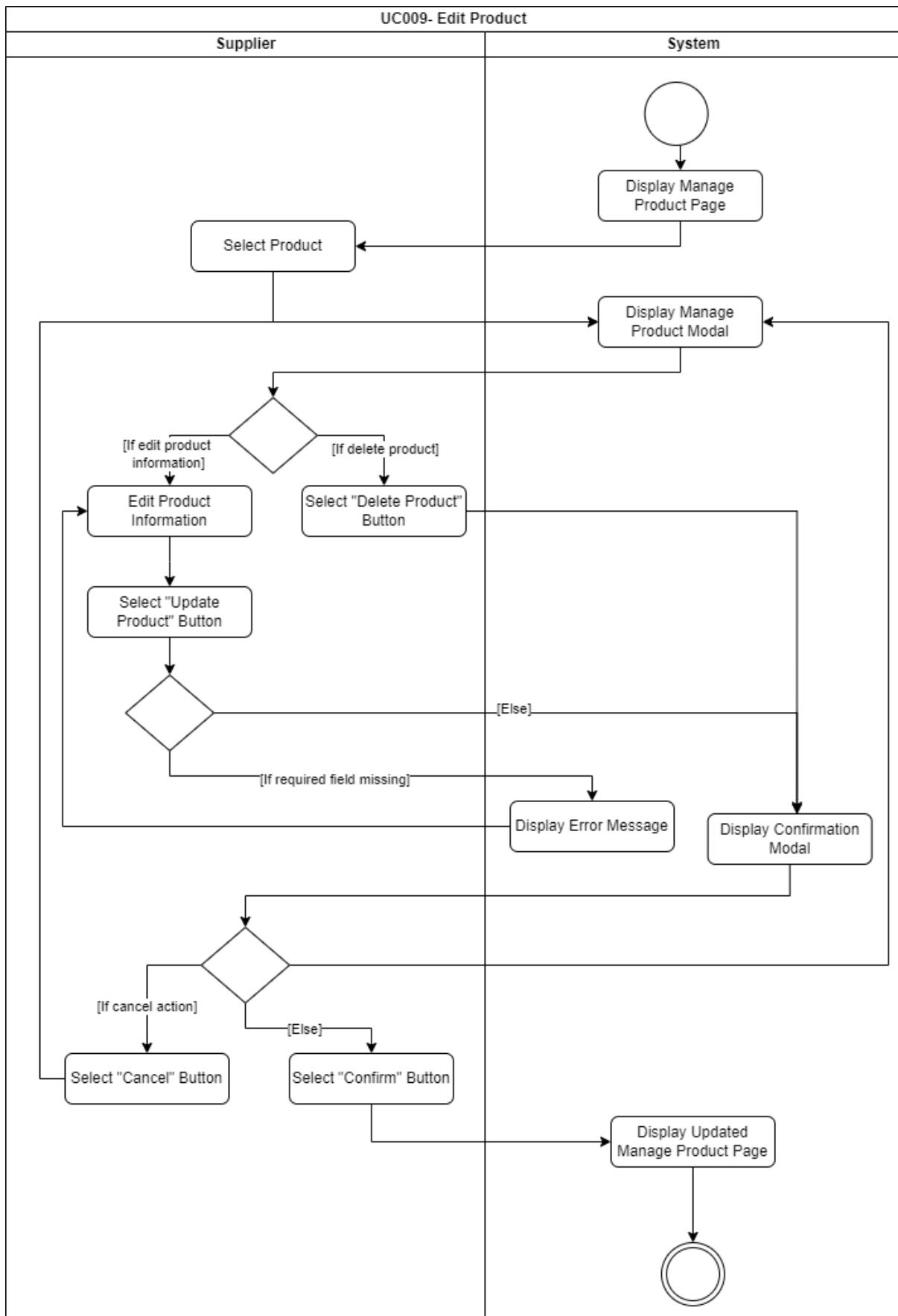


Figure 5. 29 Activity Diagram – Edit Product

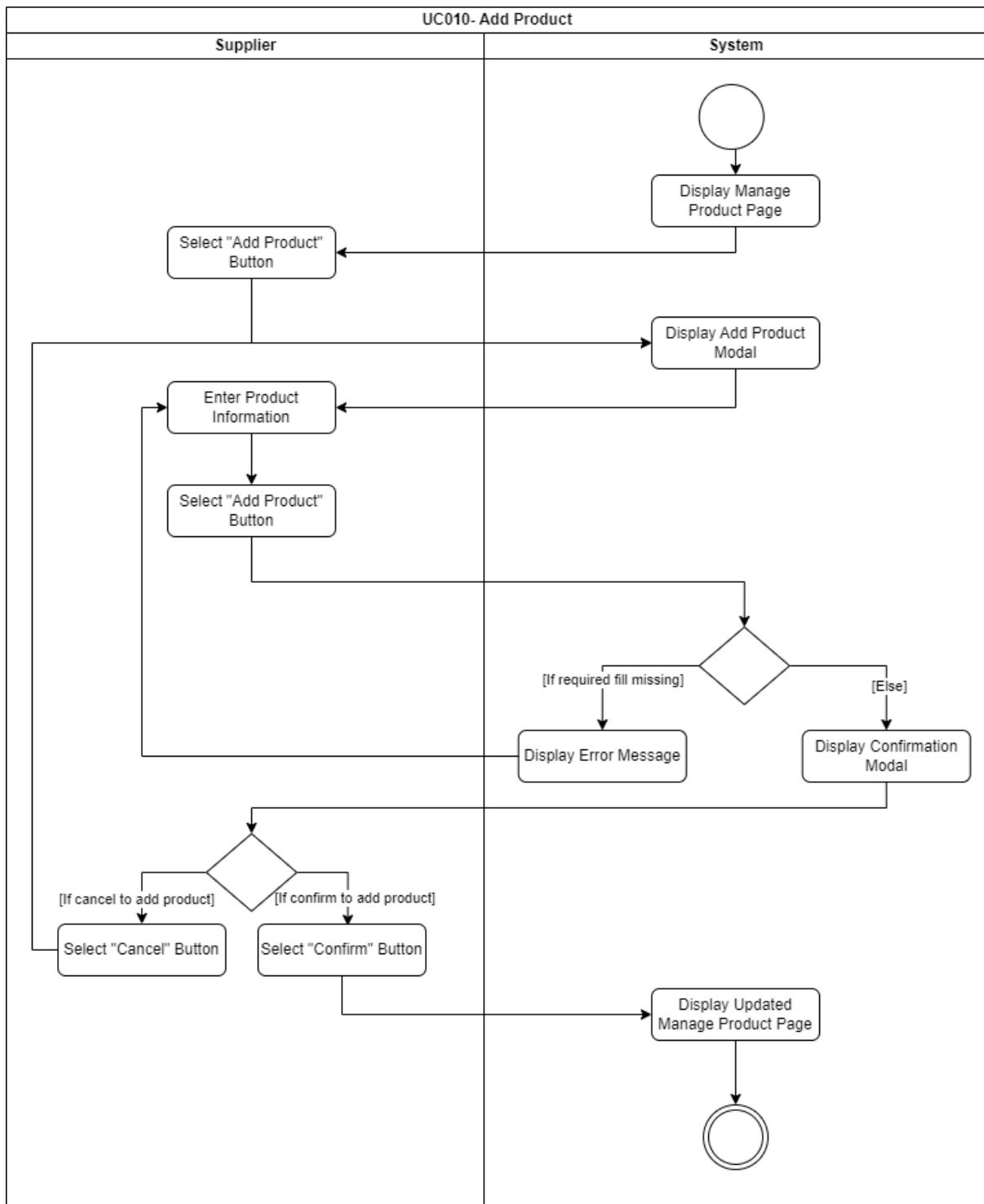


Figure 5. 30 Activity Diagram – Add Product

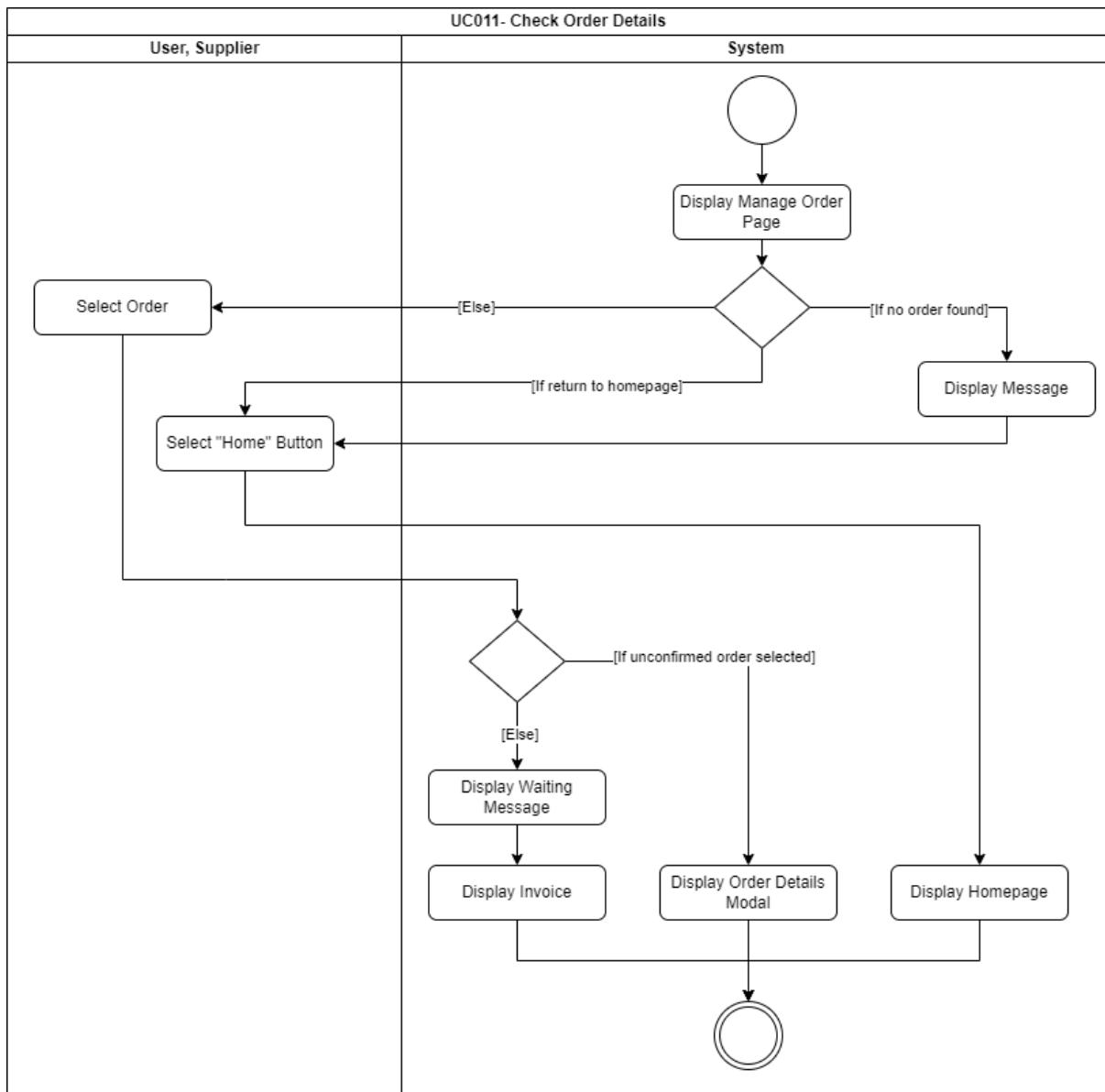


Figure 5. 31 Activity Diagram – Check Order Details

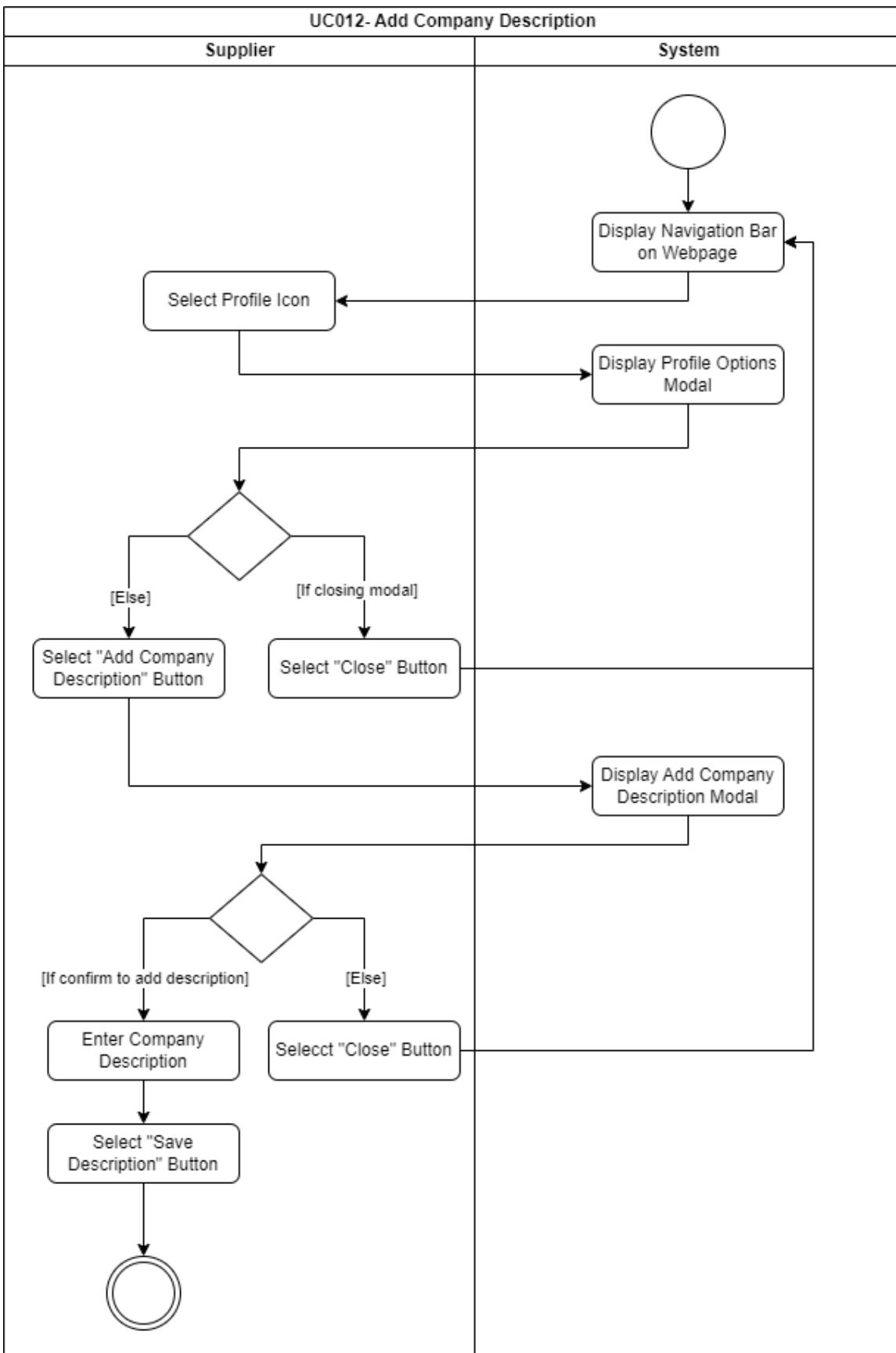


Figure 5. 32 Activity Diagram – Add Company Description

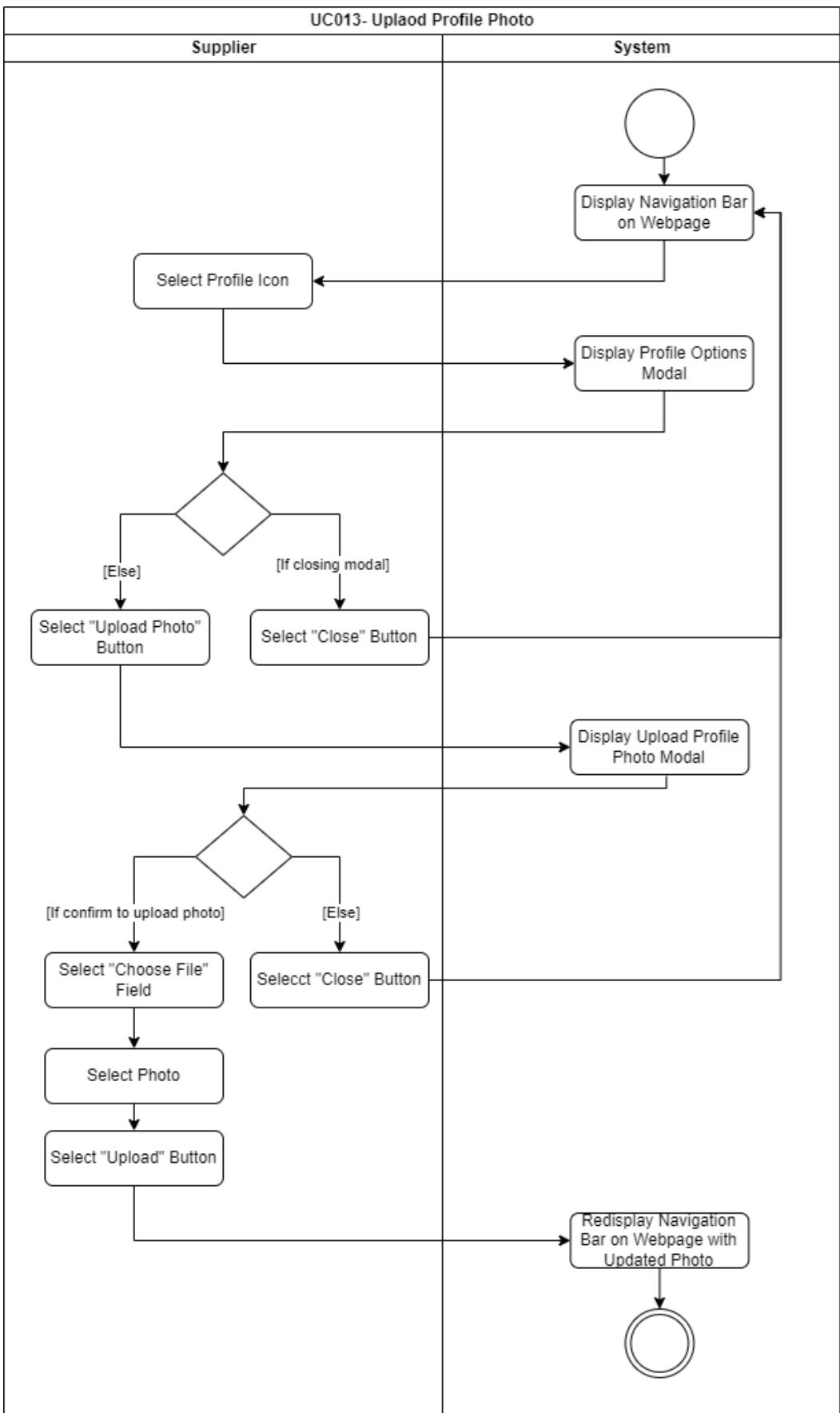


Figure 5. 33 Activity Diagram – Upload Profile Photo

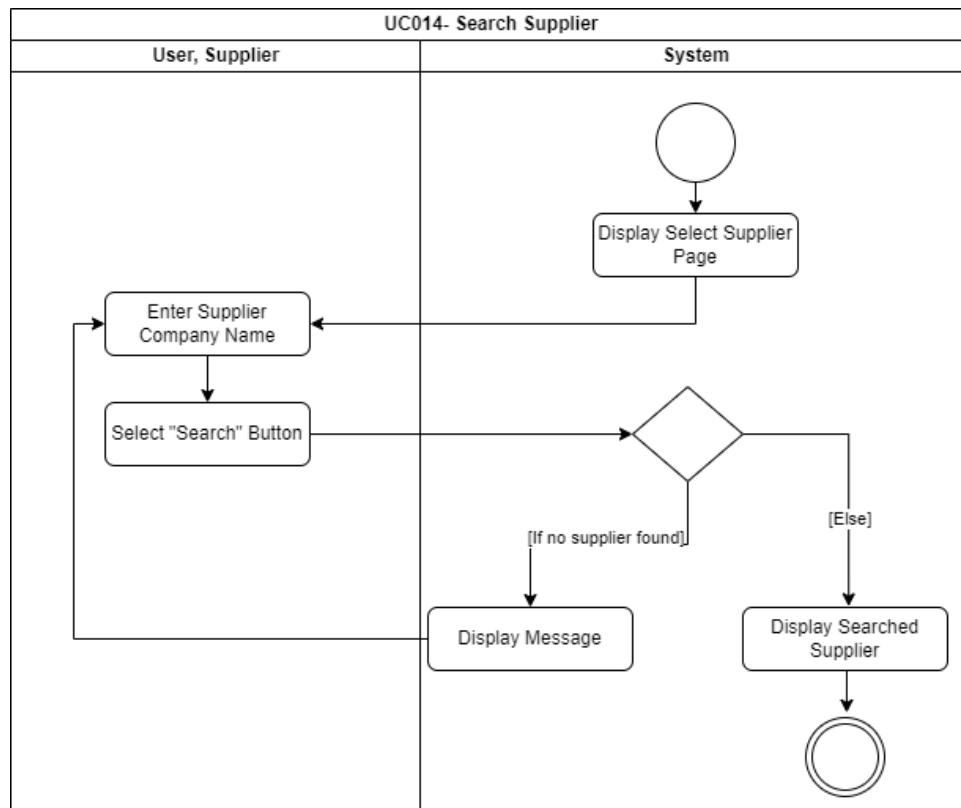


Figure 5. 34 Activity Diagram – Search Supplier

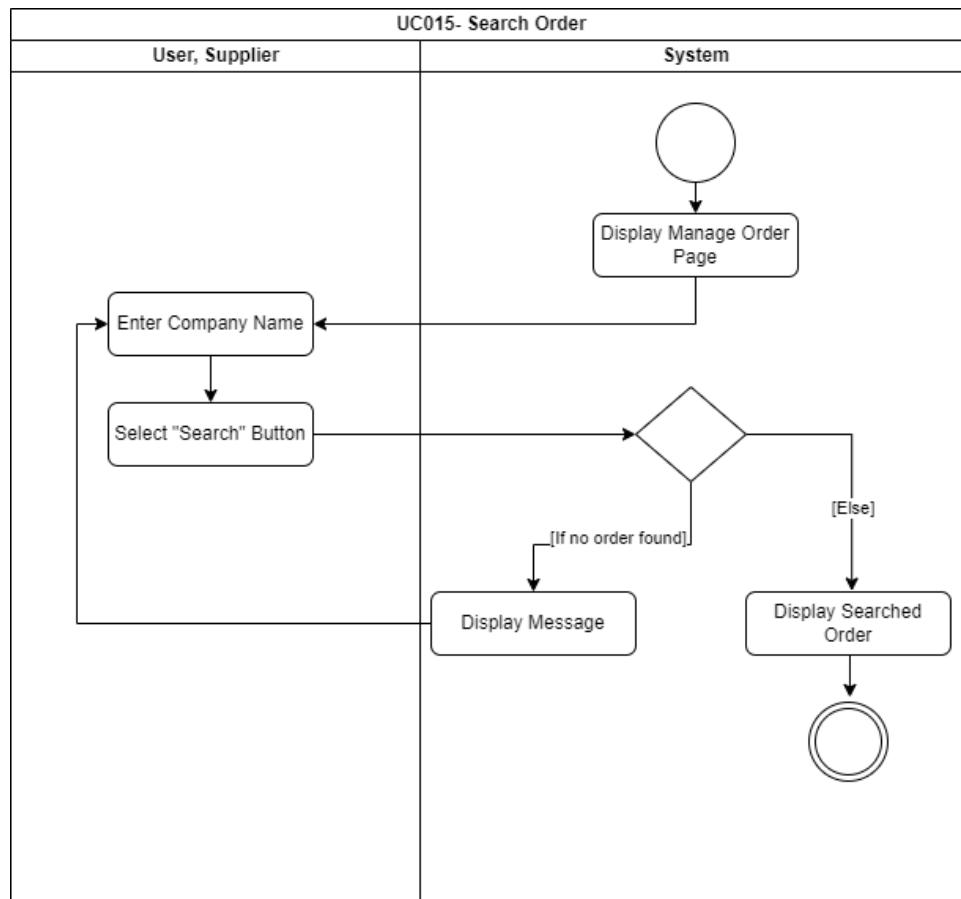


Figure 5. 35 Activity Diagram – Search Order

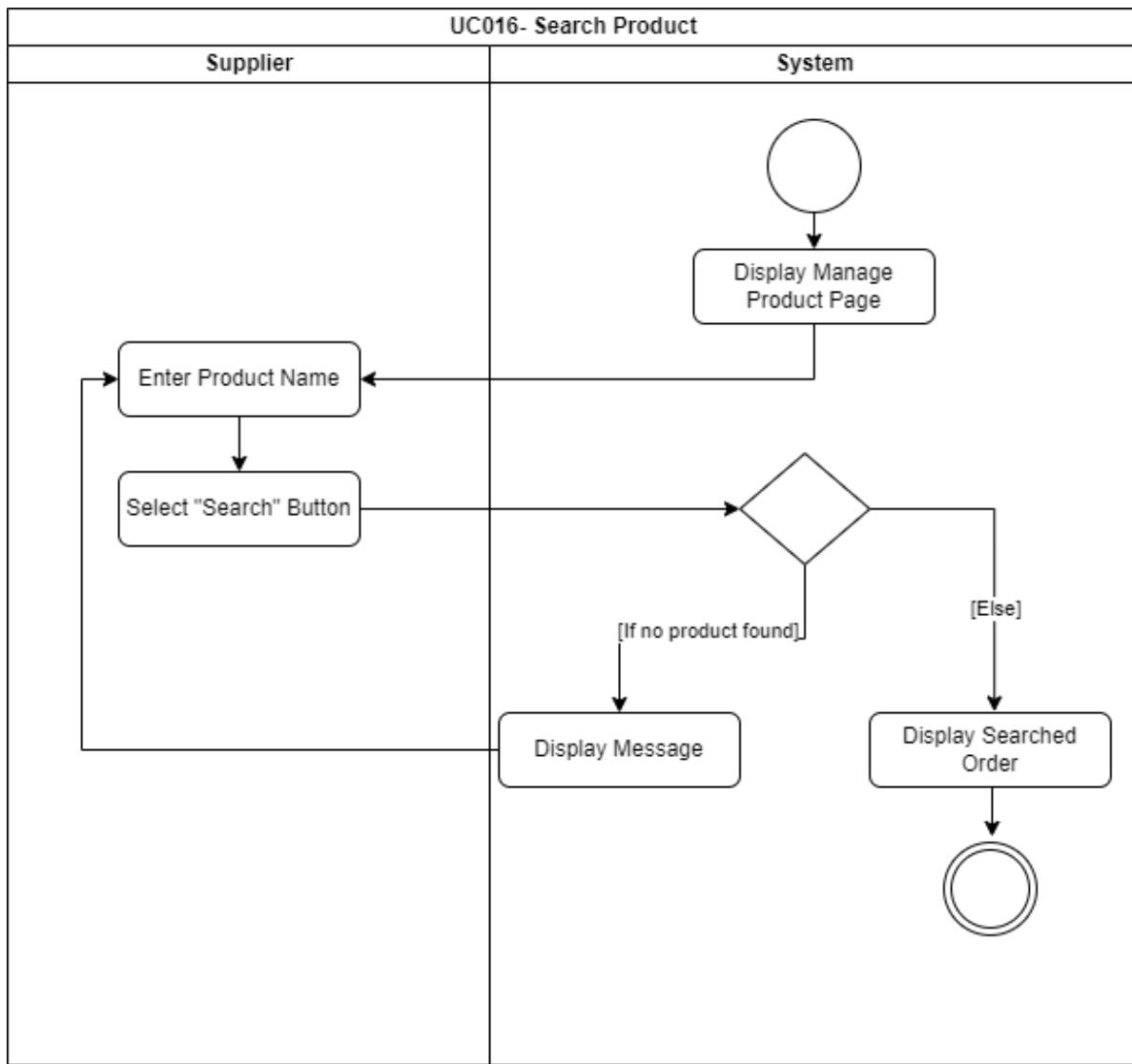


Figure 5. 36 Activity Diagram – Search Product

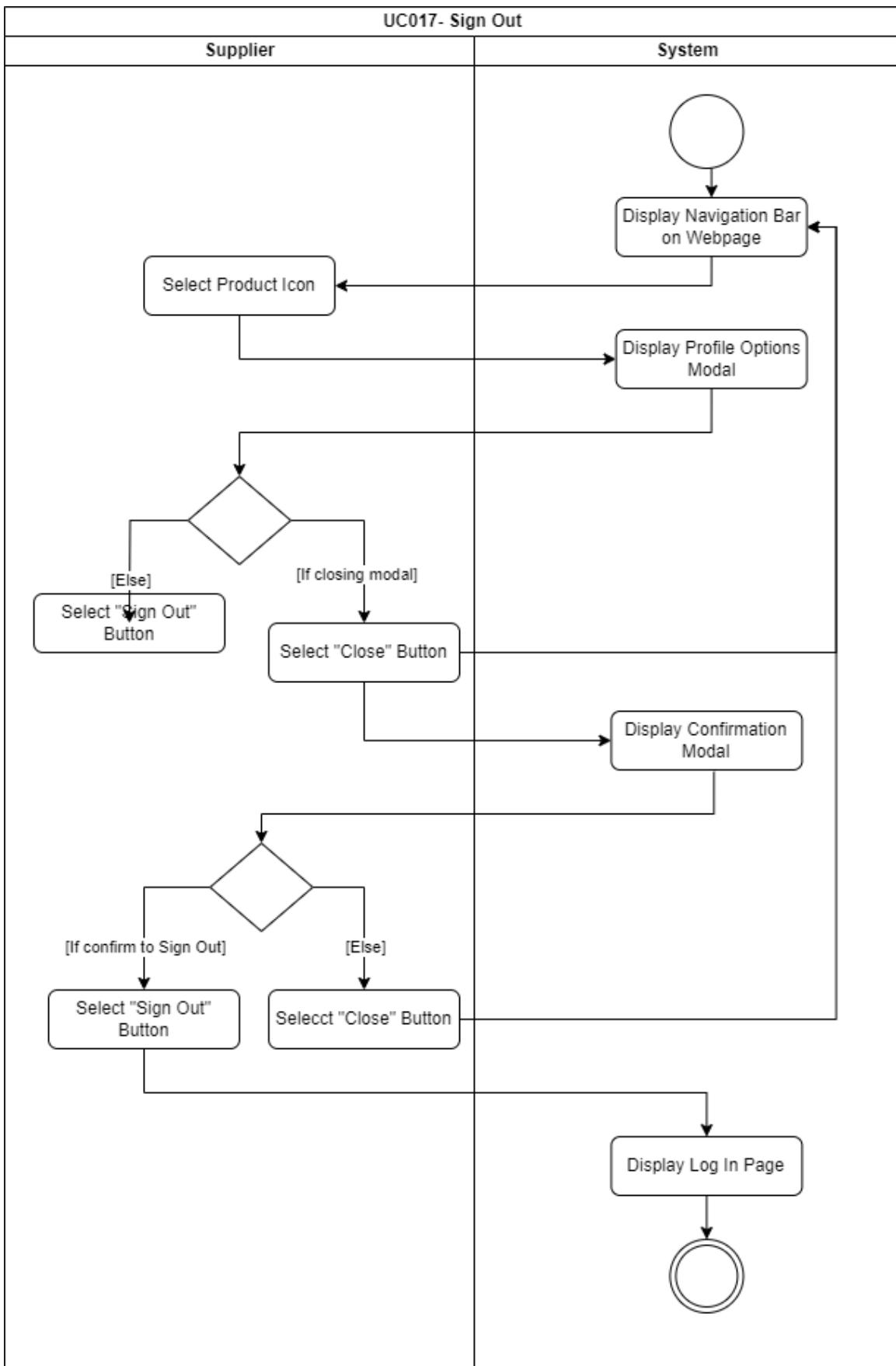


Figure 5. 37 Activity Diagram – Sign Out

5.1.7 Context Diagram

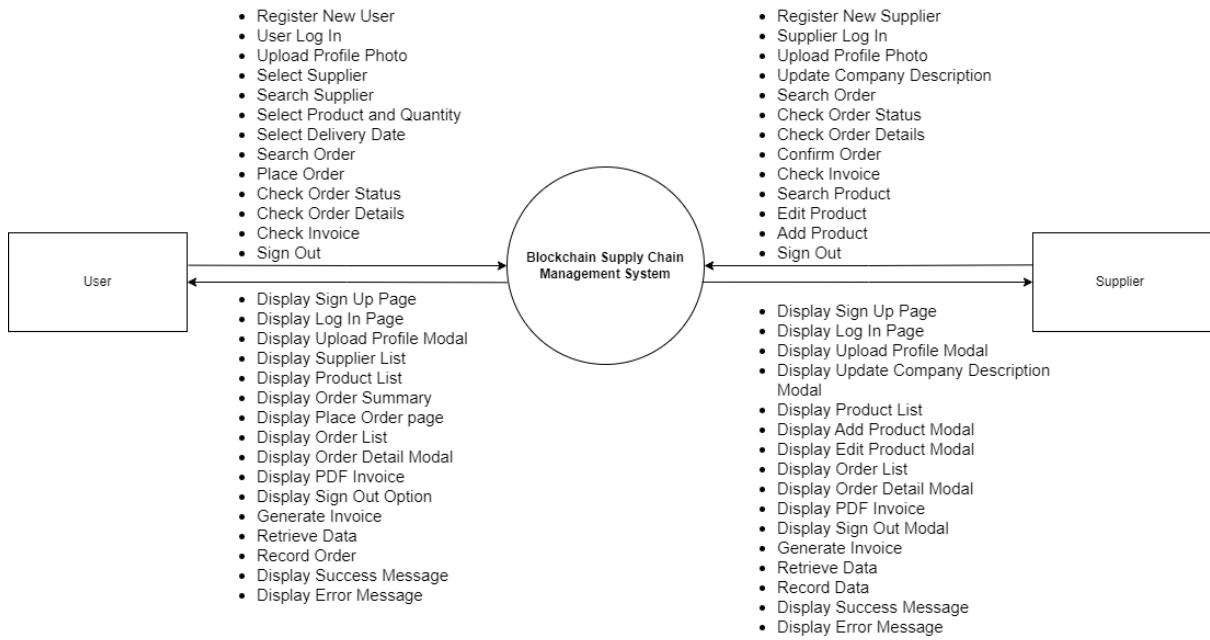


Figure 5. 38 Context Diagram – Blockchain SCMS

5.2 System Framework

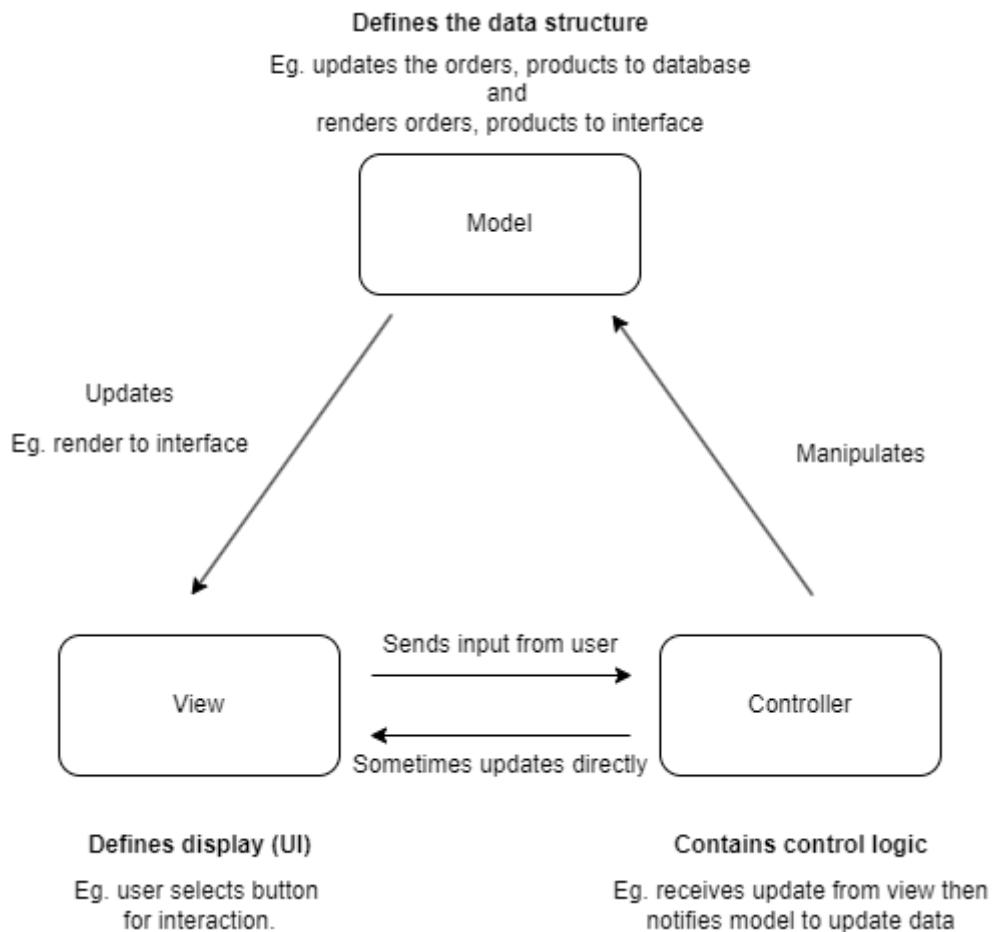


Figure 5. 39 Model View Controller Architecture Representation

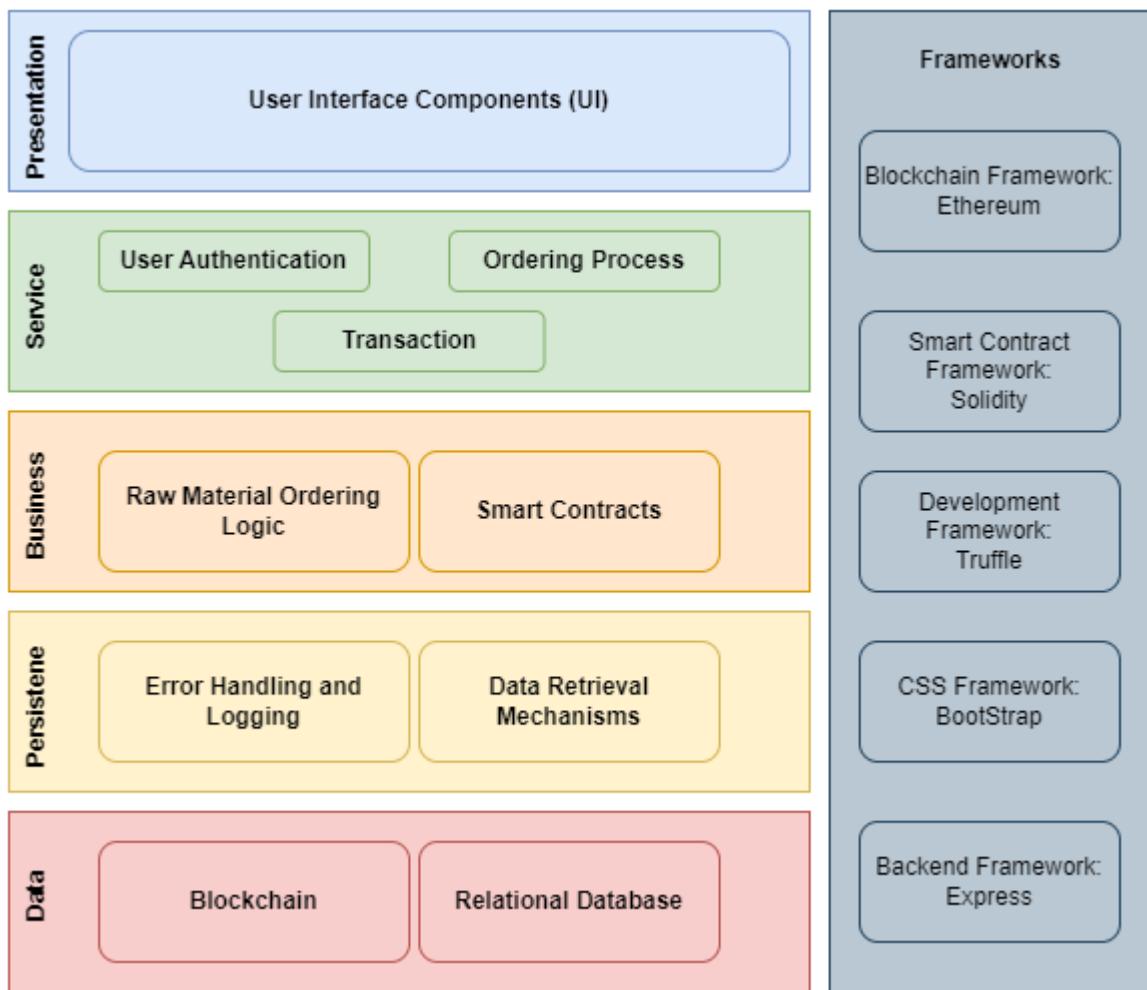


Figure 5. 40 Layered Architecture Diagram

5.3 Design Principle

5.3.1 Visibility

The system places a strong emphasis on visibility, ensuring that all functions are readily accessible and easily discoverable for the user. This is achieved through meticulous selection of icons, chosen with the aim of aligning with users' common understanding. As a result, users can navigate the interface in a clear and intuitive manner, enhancing their overall experience.

5.3.2 Consistency

In facilitating user familiarity and ease of use, particularly for first time users, consistency is paramount within the system. By following this principle, the system maintains uniformity across the interface so users can quickly grasp the system's functionality, regardless of their prior experience.

5.3.3 Constraint

Constraints within the system serve the dual purpose of ensuring error-free execution and providing clear guidance to users regarding permissible actions. For instance, the requirement to select a "delivery date" before proceeding to click the "Place Order" button establishes a predefined logic in the system's backend. This constraint not only prevents errors but also guides users through the correct sequence of actions, facilitating smooth interaction with the system while adhering to its established rules and logic.

5.3.4 Feedback

This principle of working with constraints ensures that the system maintains a proper flow of operations. For example, if a user attempts to proceed without selecting a "delivery date," the system will promptly notify them of the reason they cannot proceed, guiding them to rectify the issue. Conversely, when a user successfully completes an action, the system will provide clear notification, affirming the user's successful interaction. This approach fosters effective communication between the user and the system, enhancing user understanding and facilitating seamless navigation.

5.4 Mock Up

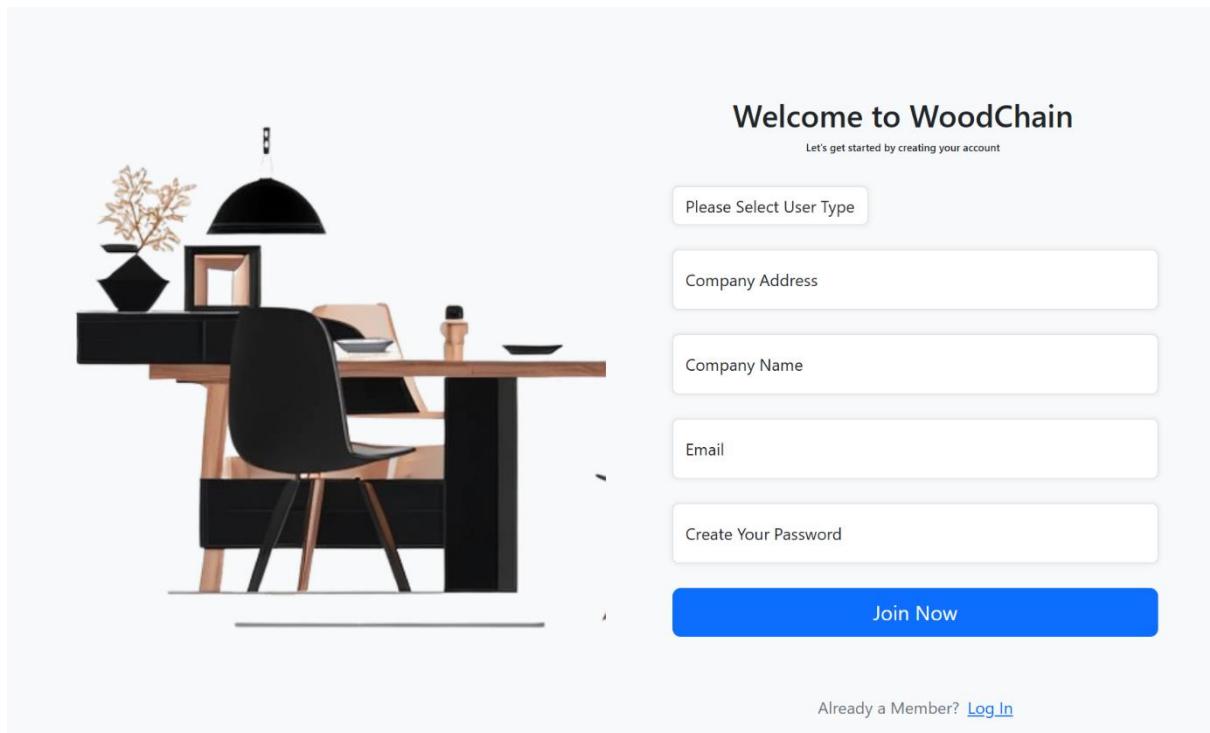


Figure 5. 41 Sign Up Page

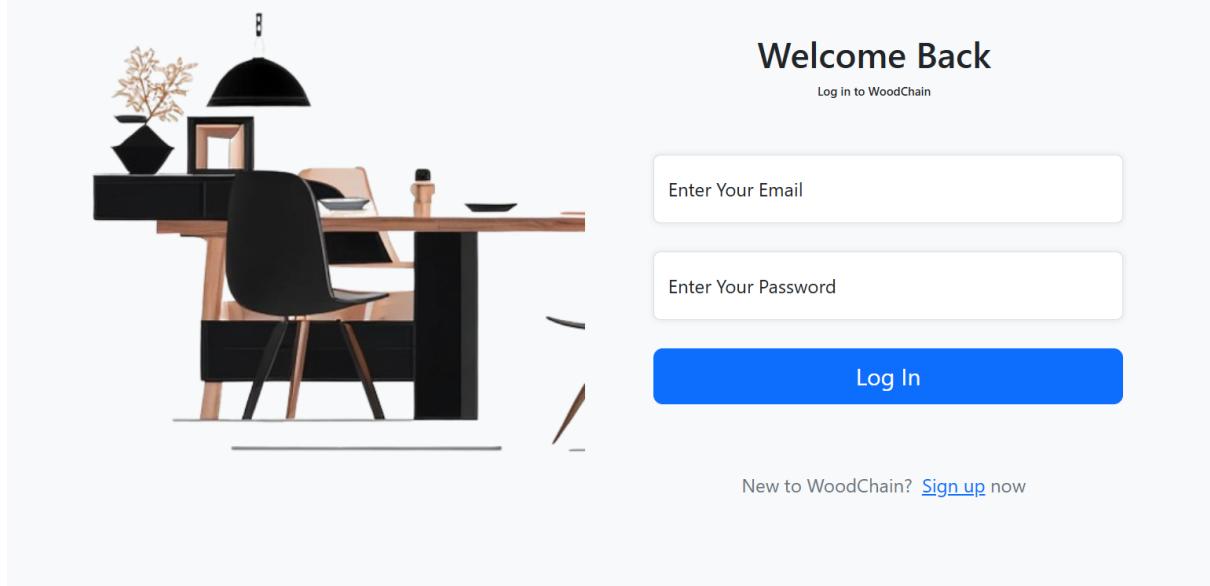


Figure 5. 42 Log In Page

Supplier List

Search supplier		Search
	Best Wood Supplier Supplier of high quality wood products	
	This is supplier 2 Supplier Description	
	This is supplier 3 Supplier Description	
	HAN CONG Sdn Bhd Supplier Description	
	A Blockchain Supply Chain Management System Supplier Description	
	This system is WoodChain Supplier Description	
	Final Year Project 2 Supplier Description	
	KINGKONG Supplier Description	
	Kim Huat Hardware Supply Supplier Description	
	What Name Else Supplier Description	

Figure 5. 43 Select Supplier Page (User Homepage)

Order Summary

Best Wood Supplier

Supplier of high quality wood products

Select product here

Pine Wood RM 79.98	2	
Proceed to Checkout		

	Oak Wood High-quality oak wood for construction RM 59.99 / each	 0 
---	--	---

	Pine Wood Sustainable pine wood RM 39.99 / each	 2 
---	--	---

Figure 5. 44 Select Product Page

Review and Submit Order

Order Details

Supplier: Best Wood Supplier

Date: 2024-07-20

Delivery Date:

Order Summary

Product ID	Product	Quantity	Price/each	Total
2	Pine Wood	2	RM 39.99	RM 79.98
Grand Total:				RM 79.98

Place Order

Figure 5. 45 Place Order Page

Manage Orders

Search order by supplier name

Orders

Order ID	Supplier Name	Date	Delivery Date	Total	Status
1	Best Wood Supplier	2024-07-20	2024-07-24	79.98	Pending
2	Best Wood Supplier	2024-07-20	2024-07-25	59.99	Pending
3	Best Wood Supplier	2024-07-20	2024-07-25	59.9	Pending
4	This is supplier 3	2024-07-20	2024-07-25	616.6	Pending
5	Best Wood Supplier	2024-07-20	2024-07-26	59.99	Pending
6	Best Wood Supplier	2024-07-20	2024-07-25	99.98	Pending
7	This is supplier 3	2024-07-20	2024-07-25	616.6	Pending
8	Best Wood Supplier	2024-07-20	2024-08-15	339.93	Pending
9	Best Wood Supplier	2024-07-20	2024-07-20	99.98	Pending
10	Best Wood Supplier	2024-07-20	2024-07-25	299.94	Pending
11	Best Wood Supplier	2024-07-20	2024-07-25	279.94	Pending
12	Best Wood Supplier	2024-07-20	2024-07-25	159.96	Pending
13	Best Wood Supplier	2024-07-20	2024-07-25	159.97	Pending
14	Best Wood Supplier	2024-07-20	2024-07-27	639.89	Pending
15	Best Wood Supplier	2024-07-20	2024-07-26	239.95	Pending
16	Best Wood Supplier	2024-07-20	2024-07-25	99.98	Pending
17	Best Wood Supplier	2024-07-20	2024-07-25	00.08	Pending

Figure 5. 46 User Manage Order Page

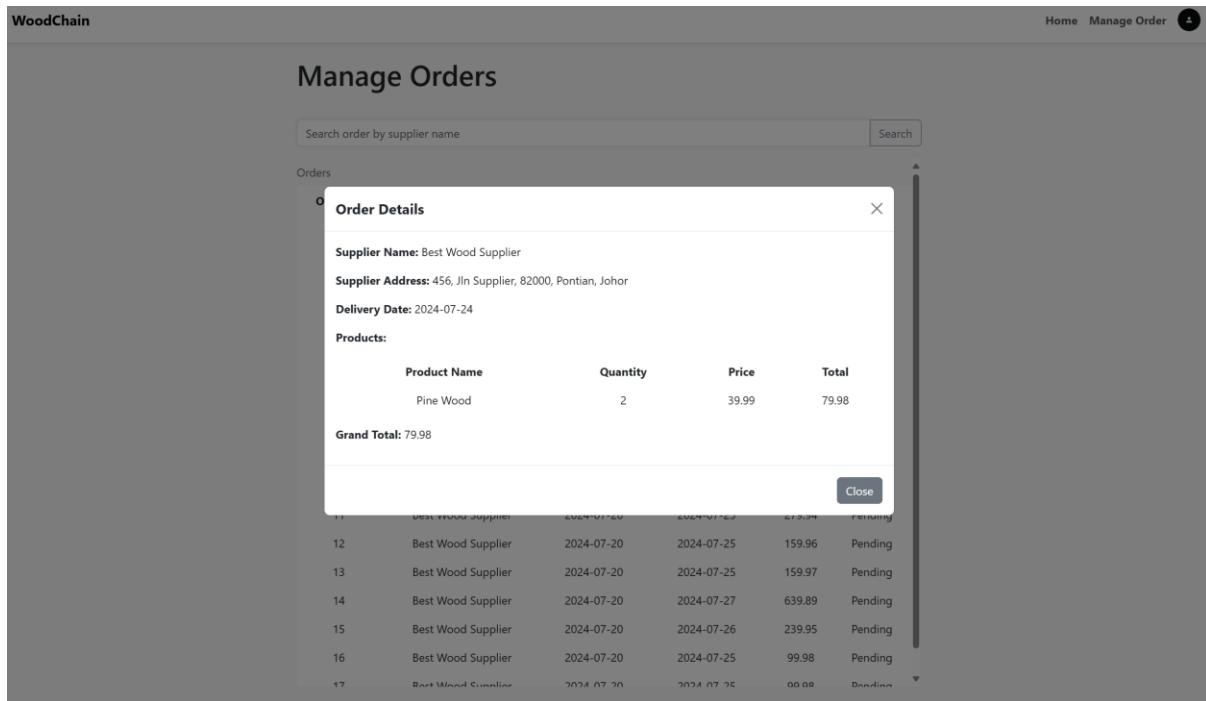


Figure 5. 47 User Side Order Details Modal

The screenshot shows the WoodChain application's interface. At the top, there is a navigation bar with links for 'Home' and 'Manage Order'. Below the navigation bar, the main title 'Product List' is displayed. A search bar is present with the placeholder 'Search product' and a 'Search' button. A 'Add Product' button is located below the search bar. The main content area displays a list of products:

Image	Product Name	Description	Price
	Oak Wood	High-quality oak wood for construction	RM 59.99 / each
	Pine Wood	Sustainable pine wood	RM 39.99 / each

Figure 5. 48 Manage Product Page (Supplier Homepage)

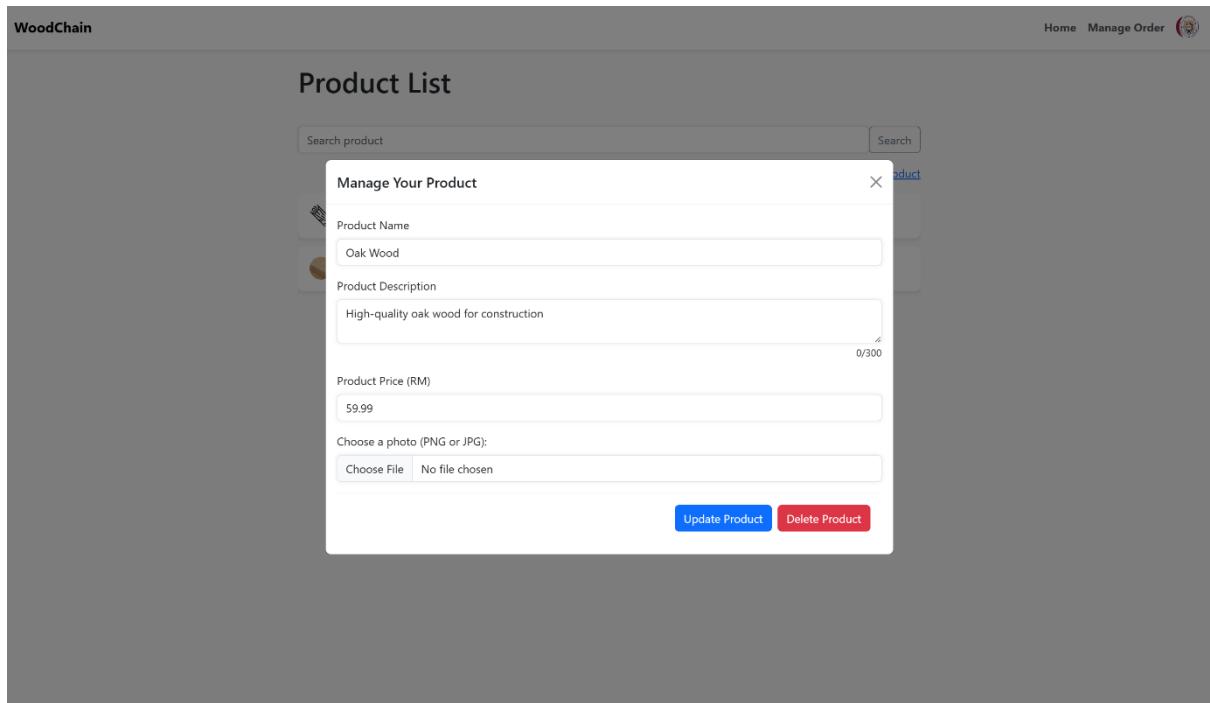


Figure 5. 49 Manage Product Modal

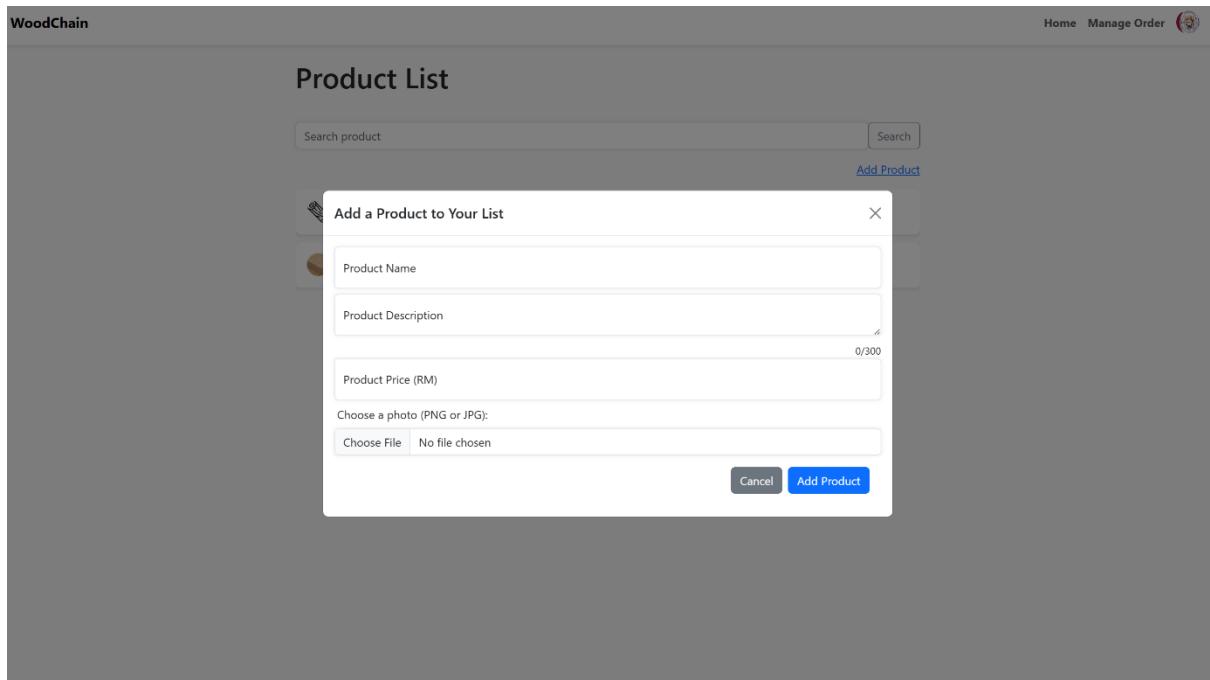


Figure 5. 50 Add Product Modal

Orders					
Order ID	Company Name	Date	Delivery Date	Total	Status
1	Apple Furniture Manufacturing	2024-07-20	2024-07-24	RM79.98	<button>Confirm Order</button>
2	Apple Furniture Manufacturing	2024-07-20	2024-07-25	RM59.99	<button>Confirm Order</button>
3	Apple Furniture Manufacturing	2024-07-20	2024-07-25	RM539.9	<button>Confirm Order</button>
5	Apple Furniture Manufacturing	2024-07-20	2024-07-26	RM59.99	<button>Confirm Order</button>
6	Apple Furniture Manufacturing	2024-07-20	2024-07-25	RM99.98	<button>Confirm Order</button>
8	Apple Furniture Manufacturing	2024-07-20	2024-08-15	RM339.93	<button>Confirm Order</button>
9	Apple Furniture Manufacturing	2024-07-20	2024-07-20	RM99.98	<button>Confirm Order</button>
10	Apple Furniture Manufacturing	2024-07-20	2024-07-25	RM299.94	<button>Confirm Order</button>
11	Apple Furniture Manufacturing	2024-07-20	2024-07-25	RM279.94	<button>Confirm Order</button>
12	Apple Furniture Manufacturing	2024-07-20	2024-07-25	RM159.96	<button>Confirm Order</button>
13	Apple Furniture Manufacturing	2024-07-20	2024-07-25	RM159.97	<button>Confirm Order</button>
14	Apple Furniture Manufacturing	2024-07-20	2024-07-27	RM639.89	<button>Confirm Order</button>

Figure 5. 51 Supplier Manage Order Page

WoodChain

Manage Orders

Search order Search

Orders

Order Details

CompanyName: Apple Furniture Manufacturing
Company Address: 123, Jln Apple, 82000, Pontian, Johor
Delivery Date: 2024-07-24

Products:

Product Name	Quantity	Price	Total
Pine Wood	2	39.99	79.98

Grand Total: 79.98

Close

11 Apple Furniture Manufacturing 2024-07-20 2024-07-25 RM279.94 Confirm Order

12 Apple Furniture Manufacturing 2024-07-20 2024-07-25 RM159.96 Confirm Order

13 Apple Furniture Manufacturing 2024-07-20 2024-07-25 RM159.97 Confirm Order

14 Apple Furniture Manufacturing 2024-07-20 2024-07-27 RM639.89 Confirm Order

Figure 5. 52 Supplier Side Order Details Modal

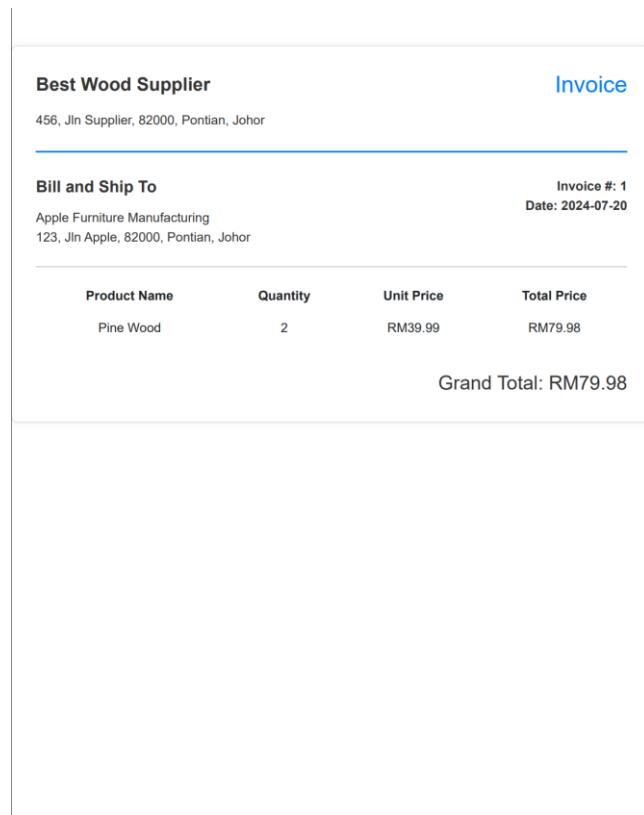


Figure 5. 53 Sample PDF Invoice

WoodChain

Supplier List

Search supplier Search

Best Wood Supplier
Supplier of high quality wood products

This is supplier
Supplier Description

This is supplier
Supplier Description

HAN CONG S
Supplier Description

A Blockchain
Supplier Description

This system is
Supplier Description

Final Year Project
Supplier Description

KINGKONG
Supplier Description

Kim Huat Hardware Supply
Supplier Description

What Name Else
Supplier Description

Profile Options

Choose an action:

Upload Photo

Sign Out

Close

Figure 5. 54 User Profile Options

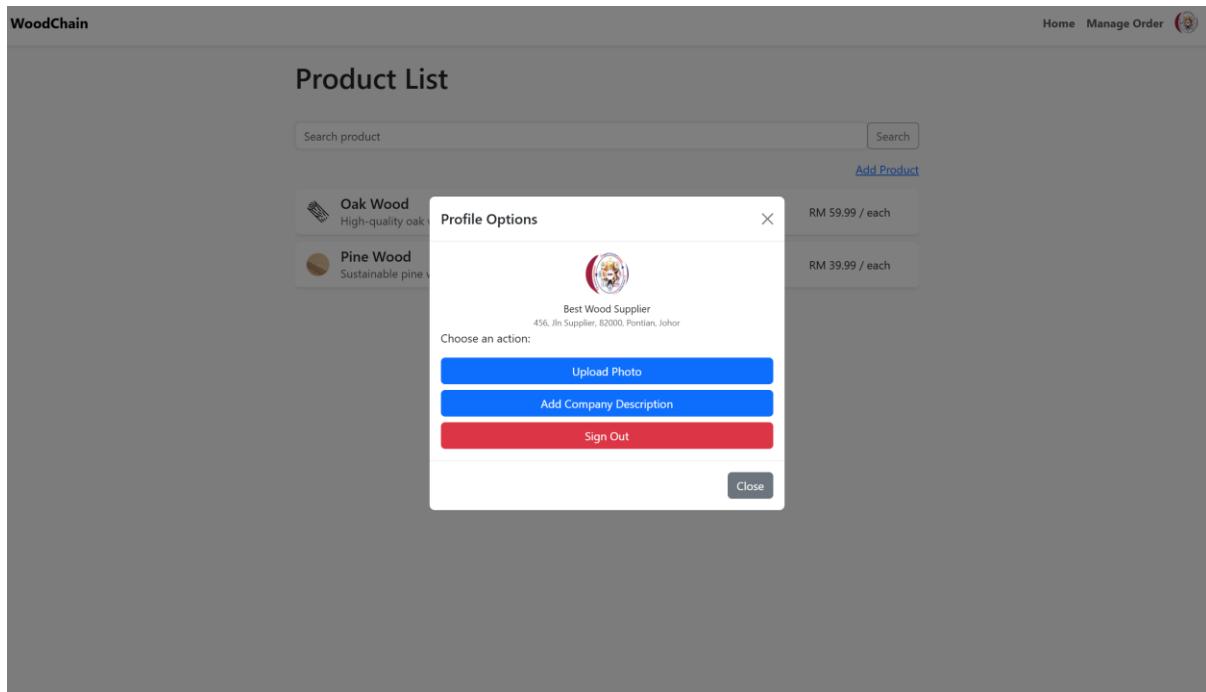


Figure 5. 55 Supplier Profile Options Modal

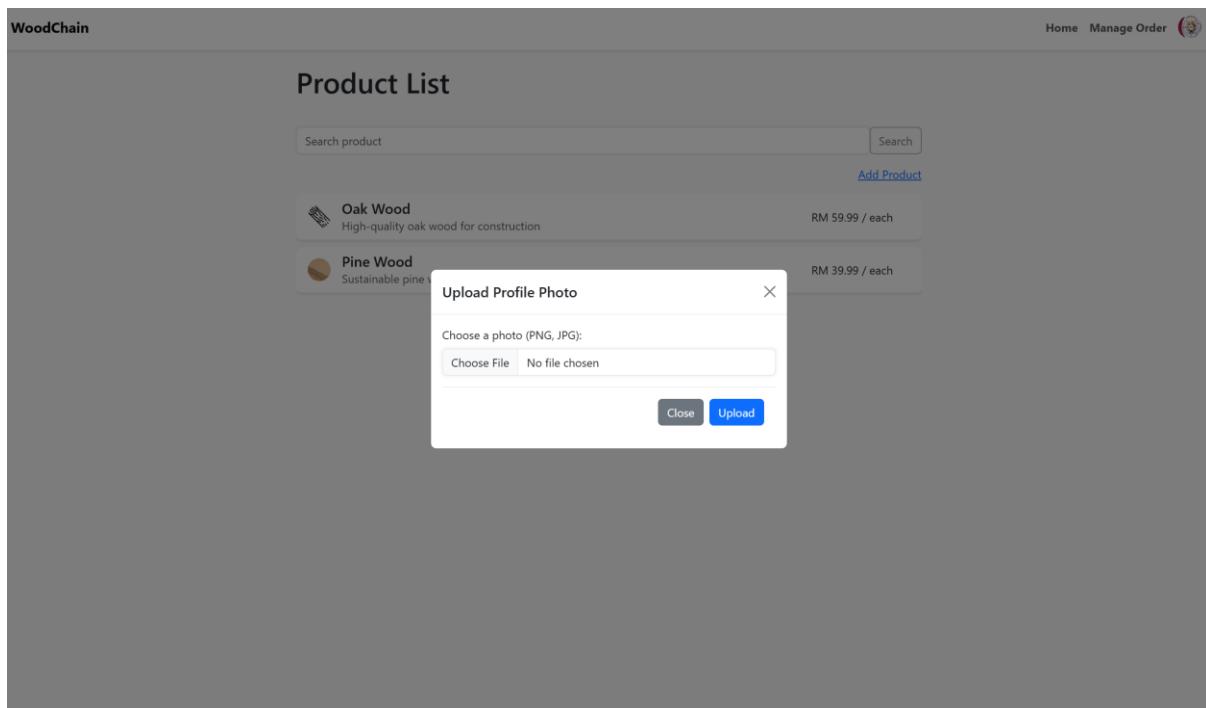


Figure 5. 56 Upload Profile Photo Modal

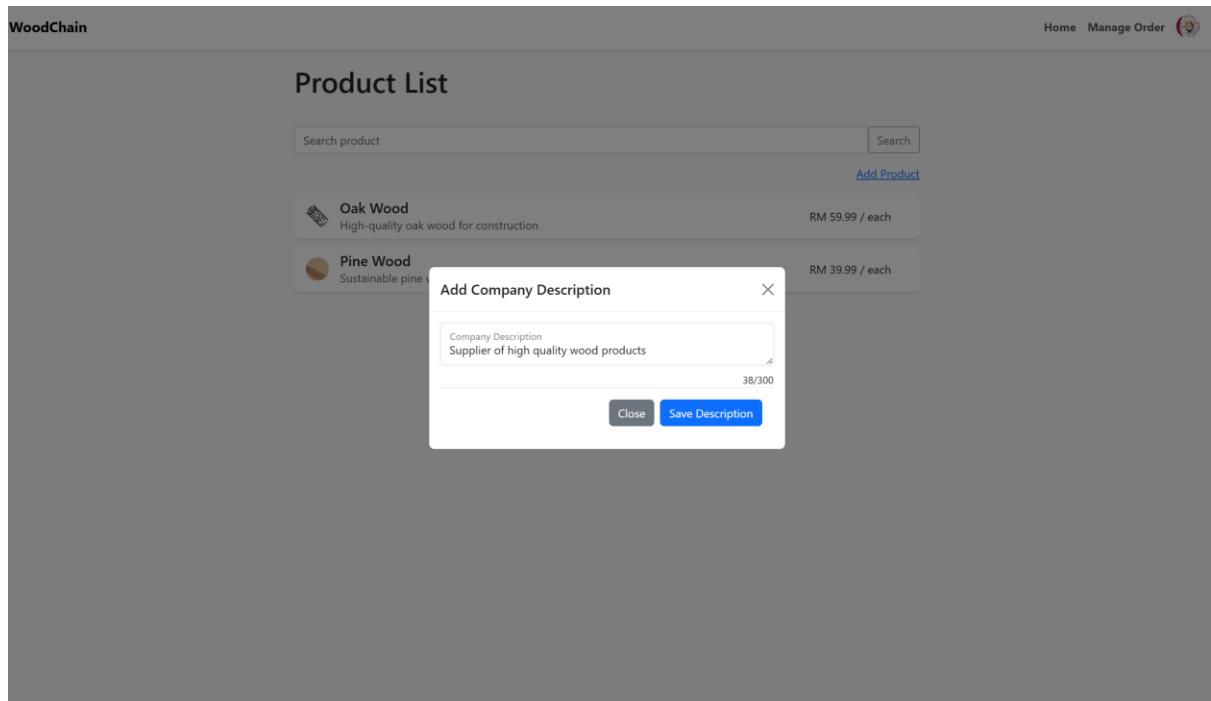


Figure 5. 57 Add Company Description Modal

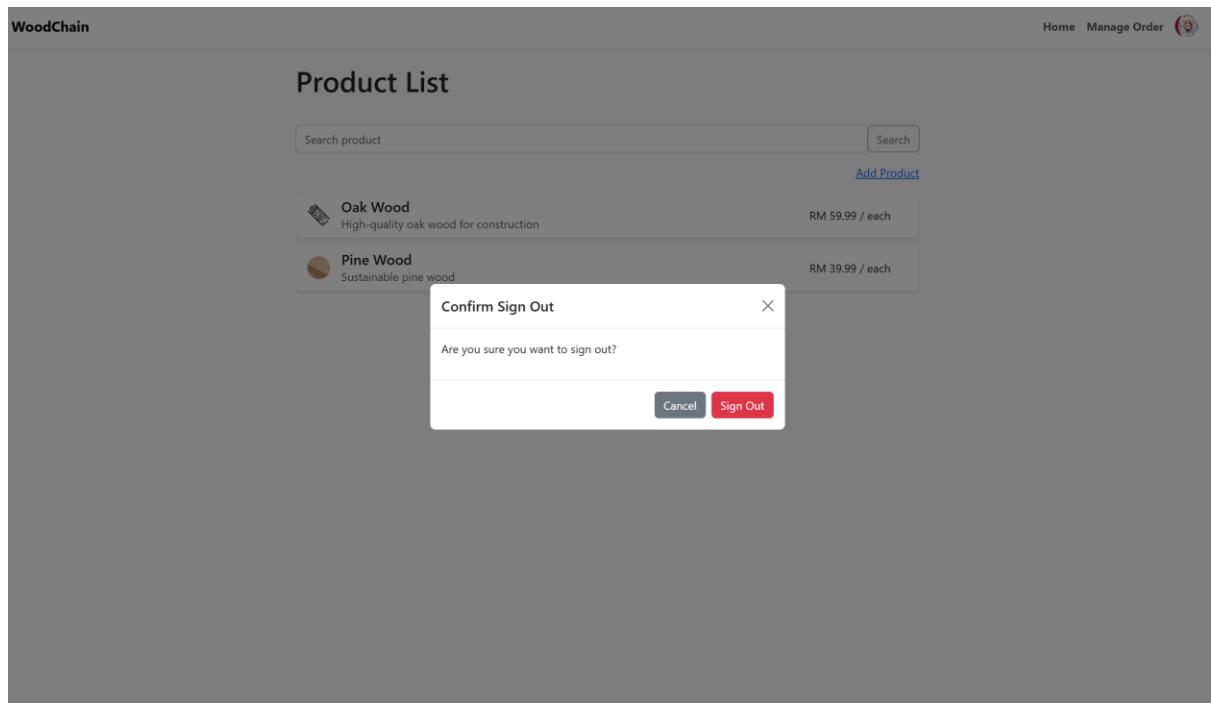


Figure 5. 58 Sign Out Confirmation Modal

Chapter 6 Implementation

This chapter aims to provide a comprehensive overview of the implementation of a Blockchain Supply Chain Management System. It includes an introduction to the directory structures from higher to lower levels, helping readers understand the application of the MVC architecture during the implementation process. Additionally, the chapter covers the implementation of functions, from basic to advanced levels, tailored to different accessibility roles to ensure clarity and readability. Lastly, the code files are commented using JSDoc notations, providing clear documentation to help readers easily follow and understand the code.

6.1 Overview of Directory Structure



Figure 6. 1 Directory Structure before Executing System

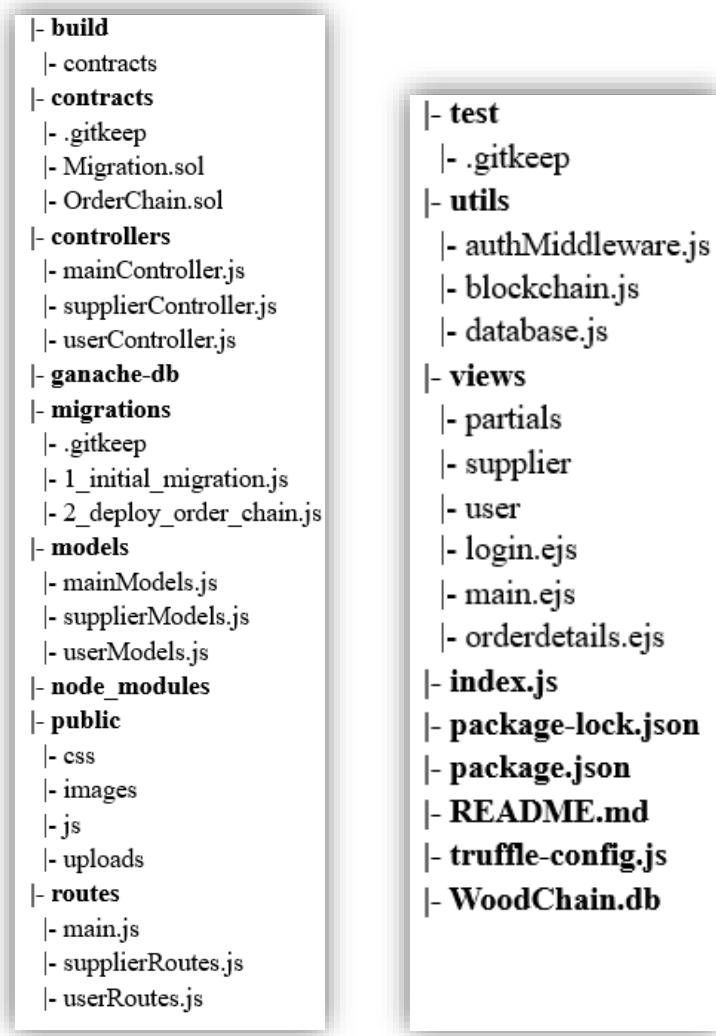


Figure 6. 2 Directory Structure after Executing System

The images above illustrate the two-tiers level of directory structure before and after executing the system. The only difference is the presence of the ‘ganache-db’ directory and the ‘WoodChain.db’ file.

6.1.1 Root Level File

Root Level File Overview	
File	Explanation
index.js	The main entry point for the system.
package-lock.json	Automatically generated file that locks the versions of the dependencies.
package.json	Contains metadata about the project, including dependencies, scripts, and other configurations.
README.md	Provides an overview of the project, instructions for setup, usage, and other relevant information.

truffle-config.js	Configuration file for truffle, specifying the network settings and other configuration for deploying smart contracts.
WoodChain.db	Database file used by the system. It stores all the data related to this system.

Table 6. 1 Root Level File Overview

6.1.2 Directories

Directories Overview	
Directory	Explanation
build	The directory contains compiled smart contracts.
contracts	The directory contains the smart contracts used in the system.
controllers	The directory contains the controller files which handle the system logic. This is part of the MVC architecture.
ganache-db	The directory is created when the Ganache blockchain simulation tool is run. This stores the local blockchain's data, including accounts, transactions, and state.
migrations	This directory contains migration scripts used to deploy and manage smart contracts on the blockchain.
models	The director contains the model files which handle the data interactions with the database. This is part of the MVC architecture.
node_modules	The directory contains all dependencies and packages which the system needed to run.
public	The directory contains static files that are served directly to the client, such as CSS, JavaScript, images, and uploaded files. The 'uploads' directory has been further categorized into 'productphoto' and 'profilephoto'.
routes	The directory contains the router files which handle the URL patterns and appropriate controller actions. This is part of the MVC architecture.
test	This directory contains test scripts to verify the functionality of the application.
utils	This directory contains the utility functions and modules that provide reusable code across the application like authentication, blockchain, and database set up.
views	This directory contains templates for rendering HTML views, which are EJS files. This is part of the MVC architecture.

Table 6. 2 Directories Overview

6.2 Overview of README.md

```
1 ## FYP 2 Project
2
3 ### GOO HAN CONG 0133677 (UEIS)
4
5 ##### System Introduction
6
7 WoodChain, a blockchain supply chain system, aims to enhance and improve supply chain management in customization furniture industry. Besides that, the information of order and order status will be recorded to blockchain to ensure the transparency of transaction on this system.
8 The functions provided and accessibility roles including as following:
9
10 - Accessibility Roles: User and Supplier
11
12 - USER:
13 - Sign Up (F001)
14 - Log In (F002)
15 - Select Supplier (F003)
16 - Place Order (F005)
17 - Check Order Status (F006)
18 - Check Invoice (F007)
19 - Check Order Details (F011)
20 - Upload Profile Photo (F013)
21 - Search Supplier (F014)
22 - Search Order (F015)
23 - Sign Out (F017)
24
25 - SUPPLIER:
26 - Sign Up (F001)
27 - Log In (F002)
28 - Check Order Status (F006)
29 - Check Invoice (F007)
30 - Confirm Order (F008)
31 - Edit Product (F009)
32 - Add Product (F010)
33 - Check Order Details (F011)
34 - Add Company Description (F012)
35 - Upload Profile Photo (F013)
36 - Search Order (F015)
37 - Search Product (F016)
38 - Sign Out (F017)
39
40 ##### Installation Requirements
41
42 - NodeJS
| - follow the install instructions at https://nodejs.org/en/
| - recommend using the latest LTS version
43 - SQLite3
| - follow the instructions at https://www.tutorialspoint.com/sqlite/sqlite\_installation.htm
44 | - Note that the latest versions of the Mac OS and Linux come with SQLite pre-installed
45
46 ##### Using this Template
47
48 To get started, you should run the syntax as following:
49
50 - Run `npm install` from the project directory to install all the node packages.
51 - Run `npm run start-ganache` to execute the blockchain.
52 - Run `npm index.js` to start serving the web app (Access via http://localhost:3000).
53
54 Test the app by browsing to the following routes:
55
56 - http://localhost:3000
57 - http://localhost:3000/login
58
59 To duplicate the project, please ensure the ganache-db, WoodChain.db and node_modules are not exist in the directory structure.
60 To reset the system (Reset everything to default):
61 Delete the directories 'ganache-db' and 'WoodChain.db' (delete database and blockchain)
62
63 ##### Languages, Framework, Tools
64
65 - System Architecture: MVC
66 - Languages: HTML, CSS, JavaScript, Solidity,
67 - Frameworks: BootStrap (CSS, JavaScript), Truffle (Smart Contract)
68 - Tools: EJS, NodeJS, Express, Web3
69
70 ##### Database Manipulation
71
72 For displaying the database in terminal
73
74 - Run 'sqlite WoodChain.db' to access the WoodChain database
75 - Run '.header on' to open the header of table when retrieve data
76 - Run '.mode column' to ensure the data can be displayed in tabular form
77 - Run 'SELECT \* FROM TABLE NAME' to retrieve and display the table in terminal
78
```

Figure 6. 3 Overview of 'README.md'

The 'README.md' file outlines the steps required to install essential tools such as Node.js and SQLite3, which are fundamental for running the system. It also provides detailed instructions on how to start the system, the routes to access various functionalities, and guidelines for database manipulation.

Moreover, the “README.md” file includes an introduction to the system, covering its name, functions, and accessibility roles. Each function is listed along with its corresponding functional requirement ID to ensure traceability in the documentation.

6.3 Overview of Utility Files Implementation

This section helps to understand the implementation of utility middleware and database configuration in WoodChain.

6.3.1 Authentication Middleware

As WoodChain is a web-based application, the route protection mechanism should be implemented to ensure security. This middleware ensures the system can avoid unauthorized access. It also contributes to the achievement of non-functional requirements in terms of the security in this project.

```
1  /* utils/authMiddleware.js */
2
3  // Middleware function to check if the user is authenticated
4  function isAuthenticated(req, res, next) {
5      if (req.session.user) {
6          return next(); // Session exists, continue to the next function in the middleware stack
7      }
8      req.session.message = "Please login first"; // Set a message in session
9      res.redirect('/login');
10 }
11
12
13 module.exports = isAuthenticated; // Exporting the authentication middleware to be used in other parts of the application
14
```

Figure 6. 4 Overview of ‘authMiddleware.js’

Therefore, this middleware is being used in every route handling.

```
51 /**
52  * @route POST /addProduct
53  * @desc Add a new product
54  * @access Private (Supplier only)
55  */
56 router.post('/addProduct', isAuthenticated, (req, res, next) => {
57     upload(req, res, (err) => {
58         if (err) {
59             res.status(500).render('supplier/supplierhome', { msg: err });
60         } else {
61             next();
62         }
63     });
64 }, supplierController.addProduct);
65
66
```

Figure 6. 5 Code Snippet in ‘supplierRoutes.js’

For example, figure 6.5 shows a code snippet from ‘supplierRoutes.js’. It applies the ‘isAuthenticated’ middleware to every route, ensuring that any attempt to manipulate routes for unauthorized access will be rejected. Unauthorized users will be redirected to the login page with an error message.

6.3.2 Database Configuration

The database.js is a file that is primarily responsible for database initialization, connection, and default data insertion. The chosen database is SQLite database, a self-contained, serverless, and zero-configuration SQL database engine. It is suitable for small to medium-sized applications.

The code can be break down into 3 parts, which are database connection, database initialization, and default data insertion.

6.3.2.1 Database Connection

```
1  /*database.js*/
2
3  const sqlite3 = require('sqlite3').verbose(); // Import the sqlite3 library and enable verbose mode for detailed error logging and debugging
4
5
6  let db = new sqlite3.Database('../WoodChain.db', sqlite3.OPEN_READWRITE | sqlite3.OPEN_CREATE, (err) => {
7    if (err) {
8      console.error('Error when creating the database', err);
9    } else {
10      console.log('Database connected!');
11      initializeDatabase(db, () => {
12        insertDefaultUser(db);
13        insertDefaultSupplier(db);
14      });
15    }
16  });
17
```

Figure 6. 6 Database Connection in 'database.js'

This block establishes a connection to the SQLite database ‘WoodChain.db’. It attempts to open the database in read-write mode and create it if it doesn't exist. Upon a successful connection, it initializes the database schema and inserts default data.

6.3.2.2 Database Initialization

```
18  function initializeDatabase(db, callback) {
19    // Serialize ensures that queries are executed sequentially
20    db.serialize(() => {
21      db.run(`
22        CREATE TABLE IF NOT EXISTS Users (
23          userID INTEGER PRIMARY KEY AUTOINCREMENT,
24          email VARCHAR(255) UNIQUE NOT NULL,
25          password VARCHAR(255) NOT NULL,
26          userType TEXT CHECK(userType IN ('User', 'Supplier')) NOT NULL,
27          companyName VARCHAR(255),
28          companyAddress VARCHAR(255),
29          profilePhoto VARCHAR(255)
30        );
31      `);
32
33      db.run(`
34        CREATE TABLE IF NOT EXISTS Suppliers (
35          supplierID INTEGER PRIMARY KEY AUTOINCREMENT,
36          userID INTEGER UNIQUE,
37          supplierDescription TEXT,
38          FOREIGN KEY(userID) REFERENCES Users(userID)
39        );
40      `);
41
42      db.run(`
43        CREATE TABLE IF NOT EXISTS Products (
44          productID INTEGER PRIMARY KEY AUTOINCREMENT,
45          supplierID INTEGER,
46          productName VARCHAR(255),
47          productDescription TEXT,
48          productPrice DECIMAL(10, 2),
49          productImage VARCHAR(255),
50          isActive BOOLEAN DEFAULT 1,
51          FOREIGN KEY(supplierID) REFERENCES Suppliers(supplierID)
52        );
53      `);
54
55      db.run(`
56        CREATE TABLE IF NOT EXISTS Orders (
57          orderId INTEGER PRIMARY KEY AUTOINCREMENT,
58          userID INTEGER,
59          supplierID INTEGER,
60          date DATE,
61          deliveryDate DATE,
62          totalPrice DECIMAL(10, 2),
63          deliveryStatus TEXT CHECK(deliveryStatus IN ('Pending', 'Confirmed')),
64          FOREIGN KEY(userID) REFERENCES Users(userID),
65          FOREIGN KEY(supplierID) REFERENCES Suppliers(supplierID)
66        );
67      `);
68
69      db.run(`
70        CREATE TABLE IF NOT EXISTS OrderDetails (
71          orderDetailID INTEGER PRIMARY KEY AUTOINCREMENT,
72          orderId INTEGER,
73          productID INTEGER,
74          productName VARCHAR(255),
75          productDescription TEXT,
76          quantity INTEGER,
77          price DECIMAL(10, 2),
78          FOREIGN KEY(orderID) REFERENCES Orders(orderID),
79          FOREIGN KEY(productID) REFERENCES Products(productID)
80        );
81      `);
82    });
83    callback(); // Call the callback after the last table is created
84  }
85
```

Figure 6. 7 Database Initialization in 'database.js'

The database is designed exactly based on the designed relational schema and ERD. Therefore, this block creates the necessary tables such as ‘User’, ‘Suppliers’, ‘Products’, ‘Orders’, and ‘OrderDetails’. Then, the data type, default value, primary and foreign keys, and so on are all defined here. In this code, SQL is being used together with JavaScript.

6.3.2.3 Default Data Insertion

```

106 // Insert default user and supplier when the database is created or reset for system testing need
107 function insertDefaultUser(db) {
108   const query = `INSERT INTO Users (email, password, userType, companyName, companyAddress) SELECT ?, ?, ?, ? WHERE NOT EXISTS (SELECT 1 FROM Users WHERE email = ?)`;
109   db.run(query, ['user@example.com', '$2a$10$ShbcPtmE3l0kk6.s.5fuZK/N.9d7712g8n806G7CVCAeProq', 'User', 'Apple Furniture Manufacturing', '123, Jln Apple, 82000, Pontian, Johor', 'user@example.com'], function(err) {
110     if (err) {
111       console.error('Error inserting default user:', err);
112     } else if (this.changes === 0) {
113       console.log('Default user already exists');
114     } else {
115       console.log('Default user inserted successfully');
116     }
117   });
118 }

```

Figure 6. 8 Default User Insertion in 'database.js'

```

100  // Insert the default supplier
101  function insertDefaultSupplier(db) {
102    db.serialize(() => {
103      db.run("BEGIN TRANSACTION;", function(err) {
104        if (err) {
105          console.error('Error starting transaction:', err);
106          return;
107        }
108
109        const userQuery = `
110          INSERT INTO Users (email, password, userType, companyName, companyAddress)
111          SELECT ?, ?, ?, ?
112          WHERE NOT EXISTS (SELECT 1 FROM Users WHERE email = ?);
113        `;
114
115        db.run(userQuery, [
116          'supplier@example.com', '$2a$10$w.2FpdjWpIvk7avUSLuvKey8rC40JB8mZEPFHZipEeXLLfCN3V7vI2',
117          'Supplier', 'Best Wood Supplier', '456, Jln Supplier, 82000, Pontian, Johor', 'supplier@example.com'], function(err) {
118          if (err) {
119            console.error('Error inserting default supplier:', err);
120            db.run("ROLLBACK;", function(rollbackErr) {
121              if (rollbackErr) console.error('Error during rollback:', rollbackErr);
122            });
123            return;
124          }
125
126          if (this.changes === 0) {
127            console.log('Default supplier already exists');
128            db.run("COMMIT;");
129            return;
130          }
131
132          console.log('Default supplier inserted successfully');
133          const userID = this.lastID;
134
135
136          const supplierQuery = `
137            INSERT INTO Suppliers (userID, supplierDescription)
138            SELECT ?, ?
139            WHERE NOT EXISTS (SELECT 1 FROM Suppliers WHERE userID = ?);
140          `;
141
142          db.run(supplierQuery, [userID, 'Supplier of high quality wood products', userID], function(err) {
143            if (err) {
144              console.error('Error inserting supplier details:', err);
145              db.run("ROLLBACK;");
146              return;
147            }
148
149            console.log('Supplier details inserted successfully');
150
151            const productQuery = `
152              INSERT INTO Products (supplierID, productName, productDescription, productPrice, productImage)
153              VALUES (?, ?, ?, ?, ?),
154                  (?, ?, ?, ?, ?);
155            `;
156            db.run(productQuery, [
157              1, 'Oak Wood', 'High-quality oak wood for construction', 59.99, '',
158              1, 'Pine Wood', 'Sustainable pine wood', 39.99, ''
159            ], function(err) {
160              if (err) {
161                console.error('Error inserting products:', err);
162                db.run("ROLLBACK;");
163                return;
164              }
165              console.log('Products inserted successfully');
166              db.run("COMMIT;", function(commitErr) {
167                if (commitErr) {
168                  console.error('Error committing transaction:', commitErr);
169                  return;
170                }
171                console.log('Transaction committed successfully');
172              });
173            });
174          });
175        });
176      });
177    });
178  }
179
180 //default supplier password : supplierpassword
181 //default user password: adminpassword
182

```

Figure 6. 9 Default Supplier Insertion in 'database.js'

The functions ‘insertDefaultUser’ and ‘insertDefaultSupplier’ insert a default user and supplier into system during its initial execution. This facilitates easier and faster system testing during implementation. Besides that, the default products which are associated with the default supplier are also added.

Lastly, the default passwords for the user and supplier are clearly commented in the code. If the reader intends to change the default passwords, they should hash the new passwords and save the hash value to the database to ensure accessibility during sign-in.

6.4 Overview of Blockchain Configuration

WoodChain leverages blockchain technology to securely record order information. When a user places an order and the supplier confirms it, the details are recorded and updated on the blockchain. This section provides an in-depth look at the blockchain configuration of WoodChain.

Regarding directory structure previously, there are several directories and files that are related to the blockchain configuration, which are ‘build’, ‘contracts’, ‘test’, ‘migrations’, ‘ganache-db’, ‘truffle-config.js’, and ‘blockchain.js’. Among them, ‘contracts’, ‘blockchain.js’, ‘truffle-config.js’, and ‘migrations’ are relatively important to be discussed for the understanding of blockchain configuration in WoodChain.

6.4.1 Smart Contract Implementation (‘contracts’)

This section covers the implementation of smart contracts, which define the business logic and rules for order processing and recording order information. The smart contracts in this system, including ‘Migration.sol’ and ‘OrderChain.sol’ are written in Solidity. These files are then placed within ‘contracts’ directory.

6.4.1.1 *Migration.sol*

```
1 // contracts/Migrations.sol
2
3 // SPDX-License-Identifier: MIT
4 pragma solidity ^0.8.0;
5
6 contract Migrations {
7     address public owner;
8     uint public last_completed_migration;
9
10    constructor() {
11        owner = msg.sender;
12    }
13
14    modifier restricted() {
15        require(
16            msg.sender == owner,
17            "This function is restricted to the contract's owner"
18        );
19        _;
20    }
21
22    function setCompleted(uint completed) public restricted {
23        last_completed_migration = completed;
24    }
25
26    function upgrade(address new_address) public restricted {
27        Migrations upgraded = Migrations(new_address);
28        upgraded.setCompleted(last_completed_migration);
29    }
30 }
```

Figure 6. 10 Overview of ‘Migration.sol’

This contract ensures that the deployment process is controlled, and that each migration is executed only once, in the correct order. Besides that, this also helps to keep track of which migration scripts have been run.

6.4.1.2 OrderChain.sol

```

1 // contracts/OrderChain.sol
2
3 // SPDX-License-Identifier: MIT
4 pragma solidity ^0.8.0;
5
6 contract OrderChain {
7     enum Status { Pending, Confirmed }
8
9     struct Order {
10         uint orderId;
11         address user;
12         uint supplierID;
13         uint date;
14         uint deliveryDate;
15         uint totalPrice;
16         Status status;
17     }
18
19     struct OrderDetail {
20         uint orderId;
21         uint productID;
22         string productName;
23         string productDescription;
24         uint quantity;
25         uint price;
26     }
27
28     mapping(uint => Order) public orders;
29     mapping(uint => OrderDetail[]) public orderDetails;
30     uint public nextOrderId = 1;
31
32     event OrderPlaced(uint orderId, address user, uint supplierID, uint totalPrice);
33     event OrderStatusUpdated(uint orderId, Status status);
34
35     function placeOrder(
36         uint _supplierID,
37         uint _deliveryDate,
38         uint _totalPrice,
39         OrderDetail[] calldata _orderDetails
40     ) external {
41         orders[nextOrderId] = Order({
42             orderId: nextOrderId,
43             user: msg.sender,
44             supplierID: _supplierID,
45             date: block.timestamp,
46             deliveryDate: _deliveryDate,
47             totalPrice: _totalPrice,
48             status: Status.Pending
49         });
50
51         for (uint i = 0; i < _orderDetails.length; i++) {
52             orderDetails[nextOrderId].push(_orderDetails[i]);
53         }
54
55         emit OrderPlaced(nextOrderId, msg.sender, _supplierID, _totalPrice);
56         nextOrderId++;
57     }
58
59     function updateOrderStatus(uint _orderId, Status _status) external {
60         require(_orderId > 0 && _orderId < nextOrderId, "Invalid order ID");
61         Order storage order = orders[_orderId];
62         require(order.user == msg.sender, "Only the user who placed the order can update its status");
63
64         order.status = _status;
65         emit OrderStatusUpdated(_orderId, _status);
66     }
67
68     function getOrder(uint _orderId) external view returns (Order memory) {
69         require(_orderId > 0 && _orderId < nextOrderId, "Invalid order ID");
70         return orders[_orderId];
71     }
72
73     function getOrderDetails(uint _orderId) external view returns (OrderDetail[] memory) {
74         require(_orderId > 0 && _orderId < nextOrderId, "Invalid order ID");
75         return orderDetails[_orderId];
76     }
77 }
```

Figure 6. 11 Overview of ‘OrderChain.sol’

This is a smart contract designed to manage orders and details on a blockchain. It specifies the license identifier, MIT licenses at the beginning. Then it begins with an enumeration named ‘Status’, which defines the possible states of an order: ‘Pending’ and ‘Confirmed’.

Then, it defines two structs which are ‘Order’ and ‘OrderDetail’. The state variable, a variable which value is permanently stored on the blockchain, including ‘order’ and ‘OrderDetail’. In addition, it also emits events ‘OrderPlaced’ and ‘OrderStatusUpdated’ for tracking new orders and status changes.

The contract includes functions such as ‘placeOrder’, which allows users to create new orders, record details, and emit the ‘OrderPlaced’ event, and ‘updateOrderStatus’, which enables users to update the status of orders and emits the ‘OrderStatusUpdated’ event.

Lastly, the view functions ‘getOrder’ and ‘getOrderDetails’ retrieve order information and details, ensuring the validity of the order ID.

6.4.2 Network Settings for Smart Contract ('truffle-config.js')

This ‘truffle-config.js’ specifies the network settings and configurations for the deployment of smart contracts using Truffle. This is an automatically generated file during truffle initialization process.

```
1 // truffle-config.js
2
3 module.exports = {
4   networks: {
5     development: {
6       host: "127.0.0.1",
7       port: 7545,
8       network_id: "5777", // Match any network id
9     },
10   },
11   compilers: {
12     solc: {
13       version: "0.8.0", // Ensure this matches your Solidity version
14     },
15   },
16 };

```

Figure 6. 12 Code Snippet of 'truffle-config.js'

The ‘networks’ object includes the configuration settings for the development blockchain. The ‘compilers’ section ensures that Truffle uses the specific version of the Solidity compiler for ensuring compatibility and consistency with the database.

6.4.3 Smart Contract Deployment ('migrations')

The ‘migrations’ directory involves the scripts which are used to deploy and manage the smart contracts on the blockchain. It mainly helps to make sure that the order and dependencies of contracts are handled properly. This directory and ‘1_initial_migration.js’ are generated automatically since Truffle provides the mechanism. However, the following scripts like ‘2_deploy_order_chain.js’ then needs to be created manually.

```
1 /* 1_initial_migration.js */
2
3 const Migrations = artifacts.require("Migrations"); // Importing the Migrations contract artifact
4
5 module.exports = function (deployer) {
6   deployer.deploy(Migrations);
7 };

```

Figure 6. 13 Overview of '1_initial_migration.js'

```

1  /* 2_deploy_order_chain.js */
2
3  const OrderChain = artifacts.require("OrderChain"); // Importing the OrderChain contract artifact
4
5  module.exports = function(deployer) {
6    deployer.deploy(OrderChain);
7 };

```

Figure 6. 14 Overview of ‘2_deploy_order_chain.js’

The first script is also the first migration script run in WoodChain. Then, ‘2_deploy_order_chain.js’ deploys the ‘OrderChain’ contract, which contains the core business logic for managing order on this system. The second script is run after the initial migration.

6.4.4 Blockchain Interaction with WoodChain (‘blockchain.js’)

The ‘blockchain.js’ file contains the utility functions for interacting with the deployed smart contracts and blockchain. This file can be organized into two parts which are initialization and setup and exported functions for blockchain interaction. The second part will be discussed in the ‘Advanced-Level Functionalities Implementation’ section.

```

1  /* blockchain.js */
2
3  const Web3 = require('web3');
4  const contract = require('@truffle/contract');
5  const orderChainArtifact = require('../build/contracts/OrderChain.json');
6
7  const options = {
8    timeout: 30000, // ms
9    clientConfig: {
10      maxReceivedFrameSize: 100000000, // bytes
11      maxReceivedMessageSize: 100000000, // bytes
12    },
13    reconnect: {
14      auto: true,
15      delay: 5000, // ms
16      maxAttempts: 5,
17      onTimeout: false
18    }
19  };
20
21 // Initialize Web3 provider and instance
22 const provider = new Web3.providers.HttpProvider('http://localhost:7545', options);
23 const web3 = new Web3(provider);
24 web3.eth.transactionPollingTimeout = 15000; // Set transaction polling interval to 15 seconds
25
26 // Set up the contract
27 const OrderChain = contract(orderChainArtifact);
28 OrderChain.setProvider(web3.currentProvider);
29
30 /**
31 * @desc Initializes the contract by fetching the deployed network and contract instance.
32 * @returns {Promise<Contract>} - The initialized contract instance.
33 * @throws {Error} - Throws an error if the contract is not deployed on the current network.
34 */
35 async function initializeContract() {
36   const networkId = await web3.eth.net.getId();
37   const deployedNetwork = orderChainArtifact.networks[networkId];
38   if (!deployedNetwork) {
39     throw new Error('Contract not deployed on the current network');
40   }
41   return new web3.eth.Contract(
42     orderChainArtifact.abi,
43     deployedNetwork.address,
44   );
45 }
46
47 const initializedContract = initializeContract();

```

Figure 6. 15 Initialization and Setup Part in ‘blockchain.js’

This part involves importing necessary libraries and contract artifacts, configuring Web3 provider options such as timeouts and reconnection settings. Additionally, it includes the initialization of the Web3 provider and instance, the setup of the contract, and the definition of contract initialization functions.

6.5 Overview of M-V-C and Router Directories

This section focuses on discussing the way in which the ‘models’, ‘views’, ‘controllers’ and ‘routes’ are categorized. Then, the required modules and utilities that need to be imported will also be discussed. Besides that, the ‘views’ will be explored in a more detailed level for the better understanding of how partial is implemented in WoodChain.

First, the files in the ‘models’, ‘views’, and ‘controllers’ directories are all based on the accessibility roles. For instance, the ‘supplierController.js’ contains all controller functions related to the supplier. The ‘mainController.js’ thereby includes all the controller functions which are accessible by both user and supplier. The ‘models’, ‘views’, and ‘routes’ directories also use the same approach in organizing their files.

6.5.1 Importing Required Modules and Utilities

6.5.1.1 Directory ‘models’

```
1  /* mainModels.js */
2
3  const db = require('../utils/database'); // Importing the database utility for performing database operations
```

Figure 6. 16 Modules and Utilities Importing in ‘mainModels.js’

```
1  /* supplierModels.js */
2
3  const db = require('../utils/database'); // Importing the database utility for performing database operations
```

Figure 6. 17 Modules and Utilities Importing in ‘supplierModels.js’

```
1  /* userModel.js */
2
3  const db = require('../utils/database'); // Importing the database utility for performing database operations
```

Figure 6. 18 Modules and Utilities Importing in ‘userModels.js’

Since the files in ‘models’ directory are used to interact with database to perform the functions such as retrieve, insert and modifying the data. Therefore, the importing of ‘database’ utility in ‘mainModels.js’, ‘supplierModels.js’, and ‘userModels.js’ can ensure that the model can interact with the database properly.

6.5.1.2 Directory ‘controllers’

```
1  /* mainController.js */
2
3  const mainModels = require('../models/mainModels'); // Importing the mainModels object that includes all the functions
4  const bcrypt = require('bcrypt'); // Importing bcrypt for password hashing
5  const path = require('path'); // Importing path for handling and transforming file paths
6  const multer = require ('multer'); //Sets up Multer for handling profile photo uploads
7  const puppeteer = require('puppeteer'); // Importing Puppeteer for generating PDFs from HTML content
```

Figure 6. 19 Modules and Utilities Importing in ‘mainControllers.js’

```
1  /* supplierController.js */
2
3  const supplierModels = require('../models/supplierModels'); // Importing supplierModels for database operations related to suppliers
4  const path = require('path'); // Importing path for handling and transforming file paths
5  const multer = require ('multer') //Sets up Multer for handling product photo uploads
6  const blockchain = require('../utils/blockchain'); // Importing blockchain utility functions for handling blockchain interactions
```

Figure 6. 20 Modules and Utilities Importing in ‘supplierControllers.js’

```

1  /* userController.js */
2
3  const userModels = require('../models/userModels'); // Ensure this path is correct
4  const blockchain = require('../utils/blockchain'); // Importing blockchain utility functions for handling blockchain interactions

```

Figure 6. 21 Modules and Utilities Importing in ‘userControllers.js’

The figures above illustrate the necessary utilities for each of the controllers. The ‘mainModels’, ‘supplierModels’, and ‘userModels’ are imported respectively to enable the controller functions to utilize the functions within its corresponding models file to perform various tasks.

Besides that, the ‘bcrypt’ library is imported to ‘mainControllers.js’ for hashing passwords during sign-up and verifying password hashes during log in. Since users and suppliers are allowed to upload photos for purposes such as profile and product images, the ‘path’ and ‘multer’ modules are imported to handle and transform file and image paths.

Furthermore, the ‘puppeteer’ library is used for generating PDF from HTML content. This is used to meet the functional requirement of ‘Check Invoice’. Lastly, the ‘blockchain’ module is used to allow the user and supplier to interact with the blockchain.

6.5.1.3 Directory ‘routes’

```

1  /* main.js */
2
3  const express = require('express'); // Importing the Express module
4  const router = express.Router(); // Creating a new router object
5  const mainController = require('../controllers/mainController'); // Importing the mainController to handle route logic
6  const isAuthenticated = require('../utils/authMiddleware'); // Importing the mainController to handle route logic

```

Figure 6. 22 Modules and Utilities Importing in ‘mainRoutes.js’

```

1  /* supplierRoutes.js */
2
3  const express = require('express'); // Importing the Express module
4  const router = express.Router(); // Creating a new router object
5  const isAuthenticated = require('../utils/authMiddleware'); // Importing the authentication middleware to protect routes
6
7  const mainRouter = require('../main');
8  router.use('/main', isAuthenticated, mainRouter); // Using mainRouter with authentication middleware
9
10 const supplierController = require('../controllers/supplierController'); // Importing the supplierController to handle route logic
11
12 // Import upload middleware
13 const { upload } = supplierController;

```

Figure 6. 23 Modules and Utilities Importing in ‘supplierRoutes.js’

```

1  /* userRoutes.js */
2
3  const express = require('express'); // Importing the Express module
4  const router = express.Router(); // Creating a new router object
5  const isAuthenticated = require('../utils/authMiddleware'); // Importing the authentication middleware to protect routes
6
7  const userController = require('../controllers/userController'); // Importing the userController to handle route logic
8
9  // Optionally using a sub-router for main if needed elsewhere
10 const mainRouter = require('../main');
11 router.use('/main', isAuthenticated, mainRouter);

```

Figure 6. 24 Modules and Utilities Importing in ‘userRoutes.js’

The router files set up routing for supplier-and-user related endpoints in an Express application. They import necessary modules and middleware, create a new router object, and apply authentication to specific routes.

6.5.2 Implementation of Views

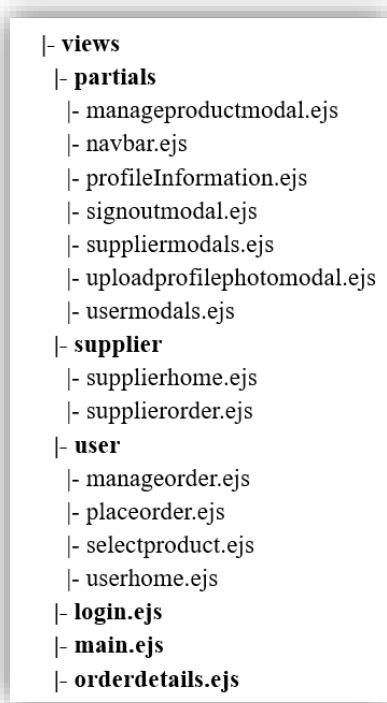


Figure 6. 25 Overview of ‘views’ Directory Structure

As mentioned above, the user interfaces are also organized based on the accessibility roles. The table below offers detailed insights into each page, explaining their purposes.

Directory ‘views’ Overview	
Directory ‘views’	
File	Explanation
login.ejs	Login page
main.ejs	Sign up page
Orderdetails.ejs	PDF invoice
Directory ‘supplier’	
File	Explanation
supplierhome.ejs	Manage order page (supplier homepage)
supplierorder.ejs	Supplier’s manage order page
Directory ‘user’	
File	Explanation
manageorder.ejs	User’s manage order page
placeorder.ejs	Place order page.
selectproduct.ejs	Select product page

userhome.ejs	Select supplier page (user homepage)
--------------	--------------------------------------

Table 6. 3 Directory ‘views’ Overview

In EJS, partial refers to a reusable EJS template that could be included in other EJS templates. It means that it can avoid code duplication by reusing common template snippets across different pages. Therefore, using partials ensures code reusability and maintainability while also providing a cleaner and more organized codebase.

Partials Overview	
Partial File	Explanation
manageproductmodal.ejs	<p>This partial includes the modals which relate to product management.</p> <ul style="list-style-type: none"> • Manage product modal • Update product confirmation modal • Confirm delete product modal
navbar.ejs	<p>All the pages contain this partial.</p> <ul style="list-style-type: none"> • Navigation bar
profileInformation.ejs	<p>This partial contains the profile information which is placed in the profile options modal</p>
signoutmodal.ejs	<p>This partial contains the sign out modal that is used in supplier and user profile options modals.</p> <ul style="list-style-type: none"> • Sign out confirmation modal
suppliermodals.ejs	<p>This partial contains the modal that are only specific to supplier.</p> <ul style="list-style-type: none"> • Supplier profile options modal • Add company description modal • Add product modal • Add product confirmation modal
uploadprofilephotomodal.ejs	<p>This partial contains the modal for uploading profile photo.</p> <ul style="list-style-type: none"> • Profile photo upload modal
usermodals.ejs	<p>This partial contains the modal that are only specific to user.</p> <ul style="list-style-type: none"> • User profile options modal

Table 6. 4 Table 6. 5 Partial Overview

6.6 Overview of index.js

```
1  /**
2   * index.js
3   * This is main app entry point
4   */
5
6 //Import necessary Modules
7 const express = require('express'); // Import the Express module for building web applications
8 const app = express(); // Initialize an Express application
9 const path = require('path'); // Import the 'path' module for handling and transforming file paths
10 var bodyParser = require('body-parser'); // Import the 'body-parser' middleware for parsing incoming request bodies
11 const session = require ('express-session'); // Import the 'express-session' middleware for managing user sessions
12 const Web3 = require('web3'); // Import the 'Web3' library for interacting with the Ethereum blockchain
13
14 //Set up session
15 app.use(session({
16   secret: 'your_secret_key',
17   resave: false,
18   saveUninitialized: true,
19   cookie: { secure: false }
20 }));
21
22 // Set view engine to EJS
23 app.set('view engine', 'ejs');
24
25 // Middleware for parsing incoming requests (Body Parser Middleware)
26 app.use(express.json());
27 app.use(express.urlencoded({ extended: true }));
28
29 //Set static folder location using path.join function
30 app.use(express.static(path.join(__dirname + '/public')));
31
32 // Exposes node_modules for client-side use of npm packages (Bootstrap)
33 app.use('/node_modules', express.static(__dirname + '/node_modules/'));
34
35 // Import database setup
36 const db = require('../utils/database');
37
38 // Define Routes
39 const mainRoutes = require('../routes/main');
40 app.use('/', mainRoutes);
41
42 //Define User Routes
43 /*
44 All the routes defined in userRoutes file
will be prefixed with /user.
45 */
46 const userRoutes = require('../routes/userRoutes');
47 app.use('/user', userRoutes);
48
49 //Define Supplier Routes
50 /*
51 All the routes defined in supplierRoutes file
will be prefixed with /supplier.
52 */
53 const supplierRoutes = require('../routes/supplierRoutes');
54 app.use('/supplier', supplierRoutes);
55
56
57 // Start the server on the specified port
58 const PORT = process.env.PORT || 3000;
59 app.listen(PORT, () => {
60   console.log(`Server is running on port ${PORT}`);
61 });


```

Figure 6. 26 Overview of 'index.js'

The 'index.js' file is the main entry point for the application. This is the reason why the system should be started with the syntax 'node index.js' in terminal panel. This file sets up and configures an Express server. The setup provides a robust foundation for the implementation of WoodChain with user and supplier functionalities, session management, and blockchain interaction capabilities.

6.6.1 Importing Required Modules and Libraries

Firstly, it imports necessary modules and libraries such as, 'express', 'path', 'body-parser', 'express-session', 'Web3'.

6.6.2 Setting Up Session Management

Then, the session management in WoodChain is set up with a secret key. It ensures that the sessions are not resaved unless modified and allows uninitialized sessions to be saved.

6.6.3 Setting Up View Engine and Middleware

The view engine is set to EJS for rendering HTML pages. Middleware is used to parse JSON and URL-encoded request bodies and to serve static files from the public directory. Additionally, the ‘node_modules’ directory is exposed for client-side use, allowing packages like Bootstrap to be included. The database setup is imported to establish a connection.

6.6.4 Defining Routes

From the perspective of routes definition, ‘mainRoutes’, ‘supplierRoutes’, and ‘userRoutes’ are imported to the main entry point of WoodChain as well. User-related routes are imported from ‘userRoutes’ and are prefixed with ‘/user’ in the URL, while supplier-related routes are imported from supplierRoutes and are prefixed with ‘/supplier’ in the URL.

6.6.5 Configuring Server

Lastly, the server is configured to listen on a defaulting port, which is port 3000.

6.7 Basic-Level Functionalities Implementation

- The system shall be able to let users register and log in an account.
- The Supply Chain Management System shall have the ability of showing the raw materials, orders and users' information.

Figure 6. 27 Basic-Level Functional Requirements

Based on the project scope, the basic-level functionalities include the functions as following:

- Shared Functionalities:
 - F001Sign Up
 - F002 Log In

This level also includes displaying the homepage, manage order page, and user information for user and supplier respectively. It focuses on the interface demonstration without manipulating data (such as inserting or deleting records in the database). However, these functionalities will be discussed alongside their related functions to ensure the documentation flows smoothly.

6.7.1 Shared Functionality

6.7.1.1 F001 Sign Up

6.7.1.1.1 Route

```
9  /**
10  * @route GET /
11  * @desc Render the Sign Up Page
12  * @access Public
13  */
14 router.get('/', (req, res) => {
15   res.render('main');
16 });
17
18 /**
19 * @route GET /main
20 * @desc Render the main (Sign Up) page with user session data
21 * @access Public
22 */
23 router.get('/main', (req, res) => {
24   res.render('main', { user: req.session.user });
25 });
26 }
```

Figure 6. 28 Route for Main Page Rendering in 'main.js'

This is a code snippet in 'main.js' router file. It helps to show the sign-up page with the URL 'http://localhost:3000/main' and 'http://localhost:3000/main'.

```

29  /**
30   * @route POST /signup
31   * @desc Handle user signup
32   * @access Public
33   */
34  router.post('/signup', mainController.signup);

```

Figure 6. 29 Route for Sign Up Request Handling in ‘main.js’

This route handles POST requests to ‘/signup’. When a user submits the sign-up form, this route triggers the ‘signup’ function in ‘mainController.js’ to process the request.

6.7.1.1.2 Model

```

7  /**
8   * @desc Finds a user by email, including supplier details if available
9   * @param {string} email - The email of the user to find
10  * @returns {Promise<Object>} - A promise that resolves with the user details, including supplierID if the user is a supplier
11  * @throws {Error} - Throws an error if there is an issue with the database query
12  */
13  findByEmail: function(email) {
14      // SQL query to find the user by email and include supplier details if available
15      return new Promise((resolve, reject) => {
16          const query = `
17              SELECT Users.*, Suppliers.supplierID
18              FROM Users
19              LEFT JOIN Suppliers ON Users.userID = Suppliers.userID
20              WHERE Users.email = ?`;
21          db.get(query, [email], (err, row) => {
22              if (err) {
23                  reject(err);
24              } else {
25                  resolve(row);
26              }
27          });
28      });
29  },
30 }

```

Figure 6. 30 Model for User Confidential Validation in ‘mainModels.js’

This is a function used to interact with the database. It retrieves user information from ‘Users’ table for authentication and authorization processes. It ensures that both regular user and supplier details are retrieved efficiently.

```

33 /**
34  * @desc Creates a new user in the database
35  * @param {Object} userData - The data of the user to be created
36  * @param {string} userData.userType - The type of the user (User or Supplier)
37  * @param {string} userData.companyAddress - The address of the user's company
38  * @param {string} userData.companyName - The name of the user's company
39  * @param {string} userData.email - The email of the user
40  * @param {string} userData.password - The password of the user
41  * @returns {Promise<number>} - A promise that resolves with the ID of the newly created user
42  * @throws {Error} - Throws an error if there is an issue with the database query
43  */
44  create: function(userData) {
45      return new Promise((resolve, reject) => {
46          const { userType, companyAddress, companyName, email, password } = userData;
47          const query = 'INSERT INTO Users (userType, companyAddress, companyName, email, password) VALUES (?, ?, ?, ?, ?)';
48          // Execute the query to insert a new user
49          db.run(query, [userType, companyAddress, companyName, email, password], function(err) {
50              if (err) reject(err);
51              resolve(this.lastID); // returns the id of the newly created user
52              console.log("User created successfully");
53          });
54      });
55  },

```

Figure 6. 31 Model for New User Creation in ‘mainModels.js’

The code snippet in figure 6.31 is designed to create a new user in the database when a new user signs up. It inserts data such as ‘userType’, ‘companyAddress’, ‘companyName’, ‘email’, and ‘password’ into the ‘Users’ table.

```

58  /**
59   * @desc Creates a new supplier in the database, including user details and supplier details
60   * @param {Object} supplierData - The data of the supplier to be created
61   * @param {String} supplierData.userType - The type of the user (Supplier)
62   * @param {String} supplierData.companyAddress - The address of the supplier's company
63   * @param {String} supplierData.companyName - The name of the supplier's company
64   * @param {String} supplierData.email - The email of the supplier
65   * @param {String} supplierData.password - The password of the supplier
66   * @param {String} supplierData.description - The description of the supplier
67   * @returns {Promise<number>} - A promise that resolves with the ID of the newly created supplier user
68   * @throws {Error} - Throws an error if there is an issue with the database query
69  */
70  createSupplier: function(supplierData) {
71    return new Promise((resolve, reject) => {
72      const { userType, companyAddress, companyName, email, password, description } = supplierData;
73
74      // Serialize the database operations to ensure they execute sequentially
75      db.serialize() => {
76        // Start a new transaction
77        db.run("BEGIN TRANSACTION;", (err) => {
78          if (err) {
79            console.error("Error starting transaction:", err);
80            return reject(err);
81          }
82
83          const userInsertQuery = 'INSERT INTO Users (userType, CompanyAddress, CompanyName, Email, Password) VALUES (?, ?, ?, ?, ?)';
84
85          // Insert user details
86          db.run(userInsertQuery, [userType, companyAddress, companyName, email, password], function(userErr) {
87            if (userErr) {
88              console.error("Error inserting user:", userErr);
89              db.run("ROLLBACK;", function(rollbackErr) {
90                if (rollbackErr) console.error("Error rolling back transaction:", rollbackErr);
91              });
92              return reject(userErr);
93            }
94
95            const userId = this.lastID;
96            console.log("User inserted with ID:", userId);
97
98            const supplierInsertQuery = 'INSERT INTO Suppliers (UserID, supplierDescription) VALUES (?, ?)';
99
100           // Insert supplier details
101           db.run(supplierInsertQuery, [userId, description], function(supplierErr) {
102             if (supplierErr) {
103               console.error("Error inserting supplier:", supplierErr);
104               db.run("ROLLBACK;", function(rollbackErr) {
105                 if (rollbackErr) console.error("Error rolling back transaction:", rollbackErr);
106               });
107               return reject(supplierErr);
108             }
109
110             // Commit the transaction to ensure all operations are saved
111             // Committing is required here to finalize the user and supplier creation
112             // If commit is not done, all changes made during the transaction will be discarded
113             db.run("COMMIT;", function(commitErr) {
114               if (commitErr) {
115                 console.error("Error committing transaction:", commitErr);
116                 return reject(commitErr);
117               }
118               console.log("Transaction committed successfully");
119               console.log("Supplier created successfully");
120               resolve(userId);
121             });
122           });
123         });
124       });
125     });
126   },
127 }

```

Figure 6. 32 Model for New User Creation in ‘mainModels.js’

This function is used to create a new supplier in the database, including its user details and supplier-specific details.

From a more detailed technical perspective, this function begins by extracting relevant data from the ‘supplierData’ object, such as ‘userType’, ‘companyAddress’, ‘companyName’, ‘email’, ‘password’, and ‘description’. It uses the ‘db.serialize’ method to ensure that database operations are executed sequentially, thereby maintaining the integrity of the transaction. The function initiates a database

transaction with ‘BEGIN TRANSACTION;’ and attempts to insert user details into the ‘Users’ table using an SQL query.

If an error occurs during this insertion, the transaction is rolled back, and the error is logged and rejected. Upon successful insertion, the user ID of the newly created user is logged, and the function proceeds to insert the supplier-specific details into the ‘Suppliers’ table, again using an SQL query. Any errors encountered during this step also trigger a rollback of the transaction. If both insert operations are successful, the transaction is committed with ‘COMMIT;’, ensuring that all changes are saved to the database. The successful completion of the transaction is logged, and the promise is resolved with the user ID of the newly created supplier. This structured approach ensures that new supplier accounts are created efficiently and securely, enhancing the application’s ability to manage supplier registrations.

6.7.1.1.3 Controller

```

51  /**
52  * @desc Handles user signup process
53  * @param {Object} req - Express request object
54  * @param {Object} req.body - Request body containing user details
55  * @param {String} req.body.userType - Type of user (User or Supplier)
56  * @param {String} req.body.companyAddress - Address of the user's company
57  * @param {String} req.body.companyName - Name of the user's company
58  * @param {String} req.body.email - Email of the user
59  * @param {String} req.body.password - Password of the user
60  * @param {Object} res - Express response object
61  * @returns {Promise<void>}
62  * @throws {Error} - Throws an error if there is an issue during the signup process
63  */
64 exports.signup = async (req, res) => {
65   // Log the request body for debugging purposes
66   console.log(req.body);
67
68   const { userType, companyAddress, companyName, email, password } = req.body;
69
70   if (!userType || userType === 'Please Select User Type') {
71     return res.status(400).render('main', { message: 'Please select a user type before signing up.' });
72   }
73
74   try {
75     // Check if user already exists
76     const existingUser = await mainModels.findByEmail(email); // Using mainModels to access findByEmail
77     if (existingUser) {
78       return res.status(400).render('main', { message: 'User already exists with the provided email.' });
79     }
80
81     // Hash password for security purpose
82     const hashedPassword = await bcrypt.hash(password, 10);
83
84     if (userType === 'Supplier') {
85       // Handle supplier record
86       const newSupplier = await mainModels.createSupplier({ // Using mainModels to access createSupplier
87         userType,
88         companyAddress,
89         companyName,
90         email,
91         password: hashedPassword,
92         description: 'Supplier Description' //Default description
93       });
94       // Redirect to login page with a success message
95       res.redirect('/login?success=true');
96     } else {
97       // Handle regular user signup
98       const newUser = await mainModels.create({ // Using mainModels to access create
99         userType,
100        companyAddress,
101        companyName,
102        email,
103        password: hashedPassword,
104      });
105      // Redirect to login page with a success message
106      res.redirect('/login?success=true');
107    }
108  } catch (error) {
109    console.error('Signup error:', error);
110    res.status(500).render('main', { message: 'Error during signup' });
111  }
112};

```

Figure 6. 33 Controller for Signing Up in ‘mainController.js’

This controller primarily manages the sign-up process. It determines which models to use based on different situations and decides which page to redirect to accordingly. For instance, it will first identify whether the ‘userType’ has been selected, and prompt error message on the interface. Then, this controller utilizes the ‘findByEmail’ function that has been mentioned previously to validate the credentials. If a user with the same email is found, it will render the ‘User already exists with the provided email’ to the interface.

After that, the function will hash the password using ‘bcrypt’ for security. Based on the ‘userType’, it will call either ‘createSupplier’ or ‘create’ from ‘mainModels.js’ to create a new supplier or user. Upon successful sign-up, the function redirects to login page.

Lastly, the error handling is also implemented for handling the unexpected error during system development and deployment.

6.7.1.1.4 View

```

49      <!--Title-->
50      <div class="card-body p-4 p-md-5 pb-xl-3 pt-xl-5 m-0">
51          <div class="row">
52              <div class="col-12 row = 15">
53                  <h4
54                      class="text-center"
55                      style="font-size: 30px"
56                  >
57                      Welcome to WoodChain
58                  </h4>
59                  <h5
60                      class="text-center"
61                      style="font-size: 10px"
62                  >
63                      Let's get started by creating your account
64                  </h5>
65
66                  <!-- Display error message -->
67                  <% if (typeof message !== 'undefined' && message &&
68                      message.length > 0) { %>
69                      <div
70                          class="alert alert-danger d-flex align-items-center"
71                          role="alert"
72                      >
73                          <%= message %>
74                      </div>
75                  <% } else { %> <% message = ''; %>
76                  <!-- Ensure message is always defined -->
77                  <% } %>
78                  <!-- End of Display error message -->
79                  </div>
80              </div>
81          </div>
82      <!-- End of Title -->
```

Figure 6. 34 Displaying Error Message in ‘main.ejs’

Line 66-78 in figure 6.34 represents the place where the error message is prompted.

```

10      <link
11          rel="stylesheet"
12          href="https://unpkg.com/bootstrap@5.3.2/dist/css/bootstrap.min.css"
13      />
14      <link
```

Figure 6. 35 Utilization of Bootstrap Framework in ‘main.ejs’

```

84      <!-- Credential Part -->
85      <form
86          action="/signup"
87          method="POST"
88      >
89      <div class="row gy-3 overflow-hidden p-2">
90          <div class="form-group col-md-5 mb-2">...
91          </div>
92
93          <div class="col-12">
94              <div class="form-floating mb-2">
95                  <input
96                      type="text"
97                      class="form-control"
98                      name="companyAddress"
99                      id="companyAddress"
100                     placeholder="Company Address"
101                     pattern=".+\S+.+"
102                     title="Company Address cannot be empty or only whitespace."
103                     required
104                     maxlength="255"
105                 />
106                 <label
107                     for="companyAddress"
108                     class="form-label"
109                 >Company Address</label>
110             >
111             </div>
112         </div>
113     </div>
114
115
116
117
118
119
120
121
122
123

```

Figure 6. 36 Required Fill in ‘main.ejs’

With CSS and JavaScript applied, and the input fields set to required, the interface will automatically prompt the user to complete the required fields if they are missed, without the need for additional programming.

```

202      <!-- Log In Section -->
203      <div class="row">
204          <div class="col-12">
205              <div class="d-flex justify-content-center mt-5">
206                  <span class="link-secondary me-2">
207                      >Already a Member?
208                  </span>
209                  <a
210                      href="/login"
211                      class="link-secondary text-decoration-underline text-primary"
212                      >Log In</a>
213                  >
214              </div>
215          </div>
216      </div>
217      <!--End of Log In Section -->
218

```

Figure 6. 37 Login Section in ‘main.ejs’

The ‘main.ejs’ provides a section for the user or supplier to access to ‘login.ejs’.

6.7.1.2 F002 Log In

6.7.1.2.1 Route

```
45  /**
46  * @route GET /login
47  * @desc Show the Log In page
48  * @access Public
49  */
50  router.get('/login', mainController.showLogin);
```

Figure 6. 38 Route for Login Page Rendering in ‘main.js’

This is a code snippet in ‘main.js’ router file. It helps to show the log in page with the URL ‘<http://localhost:3000/login>’.

```
37  /**
38  * @route POST /login
39  * @desc Handle user login
40  * @access Public
41  */
42  router.post('/login', mainController.login);
```

Figure 6. 39 Route for Login Request Handling in ‘main.js’

This route handles POST requests to ‘/login’. When a user submits the log in form, this route triggers the ‘login’ function in ‘mainController.js’ to process the request.

```
174 /**
175 * @desc Renders the login page, retrieving and clearing any session message
176 * @param {Object} req - Express request object
177 * @param {Object} req.session - Session object containing session data
178 * @param {string} req.session.message - Message stored in the session
179 * @param {Object} res - Express response object
180 */
181 exports.showLogin = (req, res) => {
182   const message = req.session.message; // Retrieve the message from the session
183   req.session.message = null; // Clear the message from the session after it's been retrieved
184   res.render('login', { message: message }); // Pass the message to the view
185 };
```

Figure 6. 40 Route for Login Page Message Handling in ‘main.js’

This code determines which message should appear on the login page based on specific conditions. For example, the login function in ‘mainController.js’ displays the message ‘Invalid email or password’ when authentication fails, while ‘authMiddleware.js’ shows ‘Please login first’ when a user tries to access a restricted page without logging in. This helps ensure that the appropriate message is shown at the right time, enhancing the user experience and security.

6.7.1.2.2 Controller

```
115  /**
116   * @desc Handles user login process
117   * @param {Object} req - Express request object
118   * @param {Object} req.body - Request body containing user credentials
119   * @param {String} req.body.email - Email of the user
120   * @param {String} req.body.password - Password of the user
121   * @param {Object} res - Express response object
122   * @returns {Promise<void>}
123   * @throws {Error} - Throws an error if there is an issue during the login process
124   */
125  exports.login = async (req, res) => {
126    const { email, password } = req.body;
127
128    try {
129      const user = await mainModels.findByEmail(email);
130      if (!user) {
131        return res.status(401).render('login', { message: 'Invalid email or password' });
132      }
133
134      const match = await bcrypt.compare(password, user.password);
135      if (!match) {
136        return res.status(401).render('login', { message: 'Invalid email or password' });
137      }
138
139      if (user.userType === 'Supplier' && user.supplierID) {
140        req.session.user = {
141          supplierID: user.supplierID, // Include supplierID in the session for suppliers
142          id: user.userID,
143          type: user.userType,
144          profilePhoto: user.profilePhoto,
145          companyName: user.companyName,
146          companyAddress: user.companyAddress,
147        };
148      } else {
149        req.session.user = {
150          id: user.userID,
151          type: user.userType,
152          profilePhoto: user.profilePhoto,
153          companyName: user.companyName,
154          companyAddress: user.companyAddress,
155        };
156      }
157
158      // Redirect based on user type
159      if (user.userType === 'Supplier') {
160        res.redirect('/supplier/supplierhome');
161        // console.log(req.session.user, "Welcome back"); For debug used
162      } else {
163        res.redirect('/user/userhome');
164        // console.log(req.session.user, "Welcome back"); For debug used
165      }
166    } catch (error) {
167      console.error('Login error:', error);
168      res.status(500).render('login', { message: 'Error logging in' });
169    }
170  };
171};
```

Figure 6. 41 Controller for Logging In in ‘mainController.js’

This controller is designed for the login process. The function begins by extracting the user's email and password from the request body. It then attempts to find the user in the database using the ‘findByEmail’ method from ‘mainModels.js’. If the user is not found or the email or password is invalid, the system will simply display the message 'Invalid email or password' to enhance the security of the authentication process.

If the user is successfully authenticated, the function sets up the user session. The user session is set up for both user and supplier with the same name ‘req.session.user’. However, the ‘req.session.user’ will contain different information based on user type. For suppliers, it includes the supplierID in the session, while for regular users, it stores the user information without the ‘supplierID’. The session includes details such as ‘userID’, ‘userType’, ‘profilePhoto’, ‘companyName’, and ‘companyAddress’.

Lastly, the controller here will redirect the user and supplier to their respective homepage.

6.7.1.2.3 View

The view and behaviour of the login page are generally similar to the sign-up page. It also uses the Bootstrap framework and has a consistent message container to manage the messages.

```
140      <!-- Signup Section -->
141      <div class="row">
142          <div class="col-12">
143              <div class="d-flex justify-content-center mt-5">
144                  <span class="link-secondary me-2">
145                      |>New to WoodChain?
146                  </span>
147                  <a
148                      href="/main"
149                      class="link-secondary text-decoration-underline text-primary"
150                      >Sign up</a>
151                  >
152                      <span class="link-secondary me-2"> &nbsp;now </span>
153                  </div>
154              </div>
155          </div>
156      <!-- End of Signup Section -->
```

Figure 6. 42 Sign-Up Section in ‘main.ejs’

The ‘login.ejs’ provides a section which is clickable for the user or supplier to access to ‘main.ejs’.

6.8 Intermediate-Level Functionalities Implementation

- The Supply Chain Management System shall have the ability of editing the raw materials and users' information.
- The Supply Chain Management System shall have the ability of searching suppliers, products, and orders by using search bar.
- The Supply Chain Management System shall have the ability of updating order status.
- The Supply Chain Management System shall have the ability of placing order.
- The Supply Chain Management System shall have the ability of invoice generation.

Figure 6. 43 Intermediate-Level Functional Requirements

Based on the project scope, the basic-level functionalities include the functions as following:

- Shared Functionalities:
 - F006 Check Order Status
 - F007 Check Invoice
 - F011 Check Order Details
 - F013 Upload Profile Photo
 - F015 Search Order
 - F017 Sign Out
- User Functionalities:
 - F003 Select Supplier
 - F004 Select Product
 - F005 Place Order
 - F014 Search Supplier
- Supplier Functionalities:
 - F008 Confirm Order
 - F009 Edit Product
 - F010 Add Product
 - F012 Add Company Description
 - F016 Search Product

6.8.1 Shared Functionality

6.8.1.1 F006 Check Order Status

This functionality involves displaying orders associated with specific users and suppliers. Therefore, the related router, model, view, and controller are not shared, even though they serve the same purpose.

6.8.1.1.1 Route

```
62  /**
63   * @route POST /placeorder
64   * @desc Record an order
65   * @access Private (User only)
66   */
67  router.get('/manageorder', isAuthenticated, userController.showManageOrderPage);
```

Figure 6. 44 Route for User Manage Order Page Rendering in ‘userRoutes.js’

```
92  /**
93   * @route GET /supplierorder
94   * @desc Show supplier orders
95   * @access Private (Supplier only)
96   */
97  router.get('/supplierorder', isAuthenticated, supplierController.showSupplierOrders);
```

Figure 6. 45 Route for Supplier Manage Order Page Rendering in ‘supplierRoutes.js’

Both code snippets in ‘userRoutes.js’ and ‘supplierRoutes.js’ serve the same purpose, which is dynamically rendering manage order page respectively. It helps to show the user and supplier manage order page with the URL ‘<http://localhost:3000/user/manageorder>’ and ‘<http://localhost:3000/supplier/supplierorder>’.

6.8.1.1.2 Model

```
196 /**
197  * @desc Retrieves the orders associated with a specific user from the database
198  * @param {number} userID - The ID of the user whose orders are being retrieved
199  * @returns {Promise<Array>} - A promise that resolves with an array of orders, or an empty array if no orders are found
200  * @throws {Error} - Throws an error if there is an issue with the database query
201 */
202 getOrdersByUserID: function(userID) {
203     return new Promise((resolve, reject) => {
204         // SQL query to fetch the orders associated with the user
205         const query = `
206             SELECT
207                 Orders.orderID,
208                 Orders.userID,
209                 Users.companyName AS supplierName,
210                 Orders.date,
211                 Orders.deliveryDate,
212                 Orders.totalPrice,
213                 Orders.deliveryStatus AS status
214             FROM Orders
215             INNER JOIN Suppliers ON Orders.supplierID = Suppliers.supplierID
216             INNER JOIN Users ON Suppliers.userID = Users.userID
217             WHERE Orders.userID = ?`;
218         db.all(query, [userID], (err, rows) => {
219             if (err) {
220                 console.error('Error fetching orders:', err);
221                 reject(err);
222             } else {
223                 console.log('All orders retrieved successfully.');
224                 resolve(rows);
225             }
226         });
227     });
228 },
229 }
```

Figure 6. 46 Model for Retrieving Order in ‘userModels.js’

```

203 /**
204 * @desc Retrieves the orders associated with a specific supplier from the database
205 * @param {number} supplierID - The ID of the supplier whose orders are being retrieved
206 * @returns {Promise<Array>} - A promise that resolves with an array of orders, or an empty array if no orders are found
207 * @throws {Error} - Throws an error if there is an issue with the database query
208 */
209 getOrdersBySupplierID: function(supplierID) {
210   return new Promise((resolve, reject) => {
211     // SQL query to fetch the orders associated with the supplier
212     const query = `
213       SELECT Orders.orderID, Users.companyName AS customerName, Orders.date, Orders.deliveryDate, Orders.totalPrice, Orders.deliveryStatus
214       FROM Orders
215       INNER JOIN Users ON Orders.userID = Users.userID
216       WHERE Orders.supplierID = ?
217       ORDER BY Orders.date DESC
218     `;
219     db.all(query, [supplierID], (err, orders) => {
220       if (err) {
221         console.error("Error fetching orders for supplierID:", supplierID, err);
222         reject(err);
223       } else {
224         console.log("Orders fetched for supplierID:", supplierID);
225         if (orders.length > 0) {
226           console.log('All orders retrieved successfully.');
227         } else {
228           console.log("No orders found for this supplier.");
229         }
230         resolve(orders);
231       }
232     });
233   });
234 },
235 };

```

Figure 6. 47 Model for Retrieving Order in ‘supplierModels.js’

The functions ‘getOrderByUserID’ and ‘getOrderBySupplierID’ are used to retrieve the relevant information for showing an order list in ‘manageorder.ejs’ and ‘supplierorder.ejs’.

From the user’s perspective, it retrieves the data, such as ‘orderID’, ‘userID’, ‘supplierName’, ‘date’, ‘deliveryDate’, ‘totalPrice’, and ‘deliveryStatus’ based on the user session ID. From the supplier’s perspective, it retrieves the data, such as ‘orderID’, ‘userID’, ‘customerName’, ‘date’, ‘deliveryDate’, ‘totalPrice’, and ‘deliveryStatus’ based on the supplier session ID. This mechanism makes sure that the displaying order is always associated with the user who has logged in.

For debugging purposes, the code includes syntax to indicate which user the order is being retrieved for and whether the order retrieval was successful.

6.8.1.1.3 Controller

```

213 /**
214 * @desc Retrieves the orders associated with a specific supplier from the database
215 * @param {Object} req - Express request object
216 * @param {Object} req.session - Session object containing user session data
217 * @param {Object} req.session.user - The current logged-in user's session data
218 * @param {number} req.session.user.id - The ID of the logged-in user
219 * @param {Object} res - Express response object
220 * @returns {Promise<void>}
221 * @throws {Error} - Throws an error if there is an issue fetching the user's orders from the database
222 */
223 exports.showManageOrderPage = async (req, res) => {
224   try {
225     const orders = await userModel.getOrdersByUserID(req.session.user.id);
226     res.render('user/manageorder', {
227       user: req.session.user,
228       orders: orders
229     });
230   } catch (error) {
231     console.error('Failed to fetch orders:', error);
232     res.status(500).render('error', { message: 'Failed to load orders.' });
233   }
234 };

```

Figure 6. 48 Controller for Displaying User Manage Order Page in ‘userController.js’

```

251 /**
252 * @desc Renders the supplier manage order page to show associated orders with the specific supplier
253 * @param {Object} req - Express request object
254 * @param {Object} req.session - Session object containing user session data
255 * @param {Object} req.session.user - The current logged-in user's session data
256 * @param {number} req.session.user.supplierID - The ID of the supplier
257 * @param {Object} res - Express response object
258 * @returns {Promise<void>}
259 * @throws {Error} - Throws an error if there is an issue fetching orders or supplier description from the database
260 */
261 exports.showSupplierOrders = async (req, res) => {
262   try {
263     const supplierID = req.session.user.supplierID;
264     const orders = await supplierModels.getOrdersBySupplierID(supplierID);
265     const description = await supplierModels.getSupplierDescription(supplierID);
266     res.render('supplier/supplierorder', {
267       user: req.session.user,
268       orders,
269       description: description || '', // Add default empty description
270       searchQuery: '' // Add default empty search query
271     });
272   } catch (error) {
273     console.error('Failed to fetch orders:', error);
274     res.status(500).send('Failed to load orders. Error: ' + error.message);
275   }
276 };

```

Figure 6. 49 Controller for Displaying Supplier Manage Order in ‘supplierController.js’

The functions from both controllers, ‘showManageOrderPage’ and ‘showSupplierOrders’, are used to display the relevant order information based on the ‘req.session.user’. The ‘getSupplierDescription’ function from ‘supplierModels’ is necessary to import here due to the inclusion of a navbar on this page that likely requires supplier-specific information. Additionally, the default values set for ‘description’ and ‘searchQuery’ serve purposes such as avoiding errors during page rendering and ensuring smooth user experience.

6.8.1.1.4 View

```

31   <!-- Header, Search Bar, and Orders Table Section -->
32   <div class="container">
33     <!-- Header and Search Bar -->
34     <h1 class="header sticky-top">Manage Orders</h1>
35   </div>
36   <!-- Search Order Bar -->
37   <div class="search-bar input-group mb-3 sticky-top">...
38   </div>
39   <!-- End of Search Order Bar -->
40   <!-- End of Header and Search Bar -->
41
42   <!-- Orders Table -->
43   <div class="table-container">
44     <table
45       class="table table-hover table-borderless caption-top text-center"
46     >
47       <caption>
48         Orders
49       </caption>
50       <thead>
51         <tr>
52           <th>Order ID</th>
53           <th>Supplier Name</th>
54           <th>Date</th>
55           <th>Delivery Date</th>
56           <th>Total</th>
57           <th>Status</th>
58         </tr>
59       </thead>
60       <tbody>
61         <% if (orders.length === 0) { %>
62           <tr>
63             <td
64               colspan="6"
65               class="text-center text-muted"
66             >
67               No orders found.
68             </td>
69           </tr>
70         <% } else { %> <% orders.forEach(order => { %>
71           <tr
72             onclick="generateInvoice('<%= order.orderID %>', '<%= order.status %>')"
73             style="cursor: pointer"
74           >
75             <td><%= order.orderID %></td>
76             <td><%= order.supplierName %></td>
77             <td><%= order.date %></td>
78             <td><%= order.deliveryDate %></td>
79             <td><%= order.totalPrice %></td>
80             <td><%= order.status %></td>
81           </tr>
82         <% }); %> <% } %>
83       </tbody>
84     </table>
85   </div>
86   <!-- End of Orders Table -->
87 </div>
88 <!-- End of Header, Search Bar, and Orders Table Section -->
89
90 <!-- Message Container -->
91 <div>...
92 </div>
93 <!-- End of Message Container -->
94
95 <!-- Overlay Message for Invoice Generation -->
96 <div>...
97 </div>
98 <!-- End of Overlay Message for Invoice Generation -->
99
100 <!-- Modals -->
101 <!-- Modal for Unconfirmed Orders -->
102 <div>...
103 </div>
104 <!-- End of Modal for Unconfirmed Orders -->
105
106 <!-- Partial Modals -->
107 <% include('../partials/usermodals') %> <%
108 include('../partials/signoutmodal') %> <%
109 include('../partials/uploadprofilephotomodal') %>
110 <!-- End of Partial Modals -->
111 <!-- End of Modals -->
112
113 <!-- Bootstrap JS -->
114 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
115
116 <!-- Client Side JS -->
117 <div>...
118 </div>
119 <!-- End of Client Side JS -->
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
587
588
589
589
590
591
592
593
594
595
596
597
597
598
599
599
600
601
602
603
604
605
606
607
607
608
609
609
610
611
612
613
613
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484

```

In addition, the script, ‘<https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js>’ is used to enable interactive components, handle smooth animations, manage dependencies, and handle utility functions. In a nutshell, it ensures the modals, alert messages, and dynamic button would function correctly in ‘manageorder.ejs’.

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8" />
5      <meta
6          name="viewport"
7          content="width=device-width, initial-scale=1.0"
8      />
9      <title>Supplier Manage Order</title>
10     <!-- Bootstrap CSS -->
11    <link
12        rel="stylesheet"
13        href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
14    />
15    <link
16        rel="stylesheet"
17        href="/css/public.css"
18    />
19    <link
20        rel="stylesheet"
21        href="/css/manageorder.css"
22    />
23    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
24    <style>
25        .align-center {
26            vertical-align: middle !important;
27        }
28    </style>
29  </head>
30
31  <body style="overflow: hidden">
32      <!-- Navigation Bar -->
33      <% include('../partials/navbar') %>
34      <!-- End of Navigation Bar -->
35
36      <!-- Header, Search Bar, and Orders Table Section -->
37      <div class="container">...
38      </div>
39      <!-- End of Header, Search Bar, and Orders Table Section -->
40
41      <!-- Message Container -->
42      <div>...
43      </div>
44      <!-- End of Message Container -->
45
46      <!-- Overlay Message for Invoice Generation -->
47      <div>...
48      </div>
49      <!-- End of Overlay Message for Invoice Generation -->
50
51      <!-- Modals -->
52      <!-- Confirm Order Confirmation Modal -->
53      <div>...
54      </div>
55      <!-- End of Confirm Order Confirmation Modal -->
56
57      <!-- Modal for Unconfirmed Orders -->
58      <div>...
59      </div>
60      <!-- End of Modal for Unconfirmed Orders -->
61
62      <!-- Partial Modals -->
63      <% include('../partials/suppliermodals', { description: description }) %>
64      <% include('../partials/signoutmodal') %> <%-
65      include('../partials/uploadprofilephotomodal') %>
66      <!-- End of Partial Modals -->
67      <!-- Modals -->
68
69      <!-- Bootstrap JS -->
70      <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
71
72      <!-- Client Side JS -->
73      <script>...
74      </script>
75      <!-- End of Client Side JS -->
76
77  </body>
78 </html>

```

Figure 6. 51 Overview of ‘supplierorder.ejs’

The interface of supplier’s manage order page includes the partials like ‘navbar.ejs’, ‘suppliermodals.ejs’, ‘signoutmodal.ejs’, ‘uploadprofilephotomodal’. Besides that, the other components are mostly the same with ‘manageorder.ejs’. This enhances the consistency of interface implementation in this system.

6.8.1.2 F007 Check Invoice

Both user and supplier share the same router, model, controller, and view for this functionality. However, they use different client-side JavaScript to handle the interaction.

6.8.1.2.1 Route

```
78  /**
79   * @route GET /orderdetails/:orderID
80   * @desc Show details of a specific order
81   * @access Private
82   */
83  router.get('/orderdetails/:orderID', isAuthenticated, mainController.showOrderDetails);
84
```

Figure 6. 52 Route for Invoice Rendering in ‘main.js’

This is a code snippet in ‘main.js’ router file. It helps to show the order details ‘invoice’ with the URL ‘<http://localhost:3000/orderdetails/orderID>’.

6.8.1.2.2 Model

```
153 /**
154  * @desc Retrieves the details of a specific order from the database
155  * @param {number} orderID - The ID of the order whose details are being retrieved
156  * @returns {Promise<Object>} - A promise that resolves with the order details
157  * @throws {Error} - Throws an error if there is an issue with the database query
158  */
159 getOrderDetailsById: function(orderID) {
160   return new Promise((resolve, reject) => {
161     // SQL query to fetch the details of the order
162     const query = `
163       SELECT Orders.*,
164             UserDetail.companyName AS userCompanyName,
165             UserDetail.companyAddress AS userAddress,
166             SupplierDetail.companyName AS supplierCompanyName,
167             SupplierDetail.companyAddress AS supplierAddress,
168             Suppliers.supplierDescription,
169             Orders.totalPrice
170           FROM Orders
171           JOIN Users AS UserDetail ON Orders.userID = UserDetail.userID
172           JOIN Suppliers ON Orders.supplierID = Suppliers.supplierID
173           JOIN Users AS SupplierDetail ON Suppliers.userID = SupplierDetail.userID
174           WHERE Orders.orderID = ?
175     `;
176     db.get(query, [orderID], (err, orderDetails) => {
177       if (err) {
178         console.error('Error fetching order details:', err);
179         reject(err);
180       } else {
181         resolve(orderDetails);
182       }
183     });
184   },
185 },
```

Figure 6. 53 Model for Retrieving Order Details in ‘mainModels.js’

This function ‘getOrderDetailsById’ performs SQL to retrieve the order details for a specific order. This information includes ‘userCompamnyName’, ‘userAddress’, ‘supplierCompanyName’,

‘supplierAddress’, ‘supplierDescription’, ‘totalPrice’ from the ‘Orders’ table. Therefore, the selected order can display the information that is retrieved by ‘getOrderDetailsById’ function here.

```
188  /**
189  * @desc Retrieves the products associated with a specific order from the database
190  * @param {number} orderID - The ID of the order whose products are being retrieved
191  * @returns {Promise<Array>} - A promise that resolves with an array of products, including the product
192  * @throws {Error} - Throws an error if there is an issue with the database query
193  */
194  getOrderProducts: function(orderID) {
195      return new Promise((resolve, reject) => {
196          // SQL query to fetch the products associated with the order
197          const query = `
198              SELECT OrderDetails.*, OrderDetails.productName, OrderDetails.price,
199                  (OrderDetails.price * OrderDetails.quantity) AS totalPrice
200              FROM OrderDetails
201              JOIN Products ON OrderDetails.productID = Products.productID
202              WHERE orderID = ?
203          `;
204          db.all(query, [orderID], (err, products) => {
205              if (err) {
206                  reject(err);
207              } else {
208                  resolve(products);
209              }
210          });
211      },
212  },
213
214};
```

Figure 6. 54 Model for Retrieving Order’s Product Details in ‘mainModels.js’

The ‘getOrderProducts’ also contributes to the demonstration of invoice. However, this function more focuses on the product details. For instance, it retrieves data such as ‘productName’, ‘price’, ‘quantity’ from the ‘OrderDetails’ table based on the given ‘orderID’.

6.8.1.2.3 Controller

```

227 /**
228 * @desc * Renders the order details for specific orders and generates PDF with the order details as an invoice
229 * @param {Object} req - Express request object
230 * @param {Object} req.params - Parameters from the request URL
231 * @param {number} req.params.orderID - The ID of the order to be displayed
232 * @param {Object} req.session - Session object containing user session data
233 * @param {Object} req.session.user - The current logged-in user's session data
234 * @param {string} req.session.user.type - The type of the logged-in user (User or Supplier)
235 * @param {Object} res - Express response object
236 * @returns {Promise<void>}
237 * @throws {Error} - Throws an error if there is an issue fetching the order details or generating the PDF
238 */
239 exports.showOrderDetails = async (req, res) => {
240   try {
241     const orderID = req.params.orderID;
242     const orderDetails = await mainModels.getOrderDetailsById(orderID);
243     const products = await mainModels.getOrderProducts(orderID);
244     let grandTotal = 0;
245
246     products.forEach(product => {
247       grandTotal += product.totalPrice;
248     });
249     if (orderDetails.deliveryStatus !== 'Confirmed') {
250       // Check the user type
251       if (req.session.user.type === 'Supplier') {
252         // If the user is a supplier, redirect to the supplier order page
253         return res.redirect('/supplier/supplierorder');
254       } else {
255         // If the user is not a supplier, redirect to the manage order page
256         return res.redirect('/user/manageorder');
257       }
258     }
259
260     // Render HTML using EJS template
261     res.render('orderdetails', {
262       order: orderDetails,
263       products: products,
264       grandTotal: grandTotal
265     }, async (err, html) => {
266       if (err) {
267         console.error('Render error:', err);
268         return res.status(500).send("Error rendering page.");
269       }
270
271       const browser = await puppeteer.launch({
272         headless: true, // Using headless for production
273         args: ['--no-sandbox', '--disable-setuid-sandbox'] // These arguments help with running Chrome in certain environments
274       );
275
276       try {
277         const page = await browser.newPage();
278         await page.setContent(html, { waitUntil: 'networkidle0' });
279         const pdf = await page.pdf({ format: 'A4' });
280         res.contentType('application/pdf');
281         res.send(pdf);
282       } catch (pdfError) {
283         console.error('PDF generation error:', pdfError);
284         res.status(500).send("Failed to generate PDF.");
285       } finally {
286         await browser.close();
287       }
288     });
289
290   } catch (error) {
291     console.error('Failed to fetch order details:', error);
292     res.status(500).render('error', { message: 'Failed to load order details.' });
293   }
294 }
295

```

Figure 6. 55 Controller for Displaying Invoice in ‘mainController.js’

This defines a function, ‘showOrderDetails’, in ‘mainController.js’. This function handles the process of fetching order details and generating a PDF invoice based on those details.

The function extracts the ‘orderID’ from the request parameters and uses ‘mainModels.getOrderDetailsById’ and ‘mainModels.getOrderProducts’ to fetch the order details and associated products. Besides that, it also provides the mechanism to ensure that the invoice will be generated when the order status is not ‘Confirmed’.

Then, the PDF invoice is generated using Puppeteer library. It will create a new page for rendering the PDF. The browser instance is closed in the ‘finally’ block to ensure it always closes, even if an error occurs.

6.8.1.2.4 View

```

1  <!DOCTYPE html>
2  <html lang="en">
3      <head>
4          <meta charset="UTF-8" />
5          <meta
6              name="viewport"
7              content="width=device-width, initial-scale=1.0"
8          />
9          <title>Invoice - Order ID: <%= order.orderID %></title>
10         <!-- Bootstrap CSS -->
11         <link
12             rel="stylesheet"
13             href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
14         />
15     <style>...
16     </style>
17 </head>
18
19 <body>
20     <div class="invoice-box">
21         <!-- Header -->
22         <div class="invoice-header">
23             <div>
24                 <p class="fw-bold fs-4"><%= order.supplierCompanyName %></p>
25                 <p><%= order.supplierAddress %></p>
26             </div>
27             <div class="text-end fw-bold">
28                 <h2>Invoice</h2>
29             </div>
30         </div>
31         <!-- End of Header -->
32
33         <!-- Invoice Info and Customer Info -->
34         <div class="row mb-4">
35             <div class="col-md-6">
36                 <div class="info-section">
37                     <h5 class="fw-bold">Bill and Ship To</h5>
38                     <p><%= order.userCompanyName %></p>
39                     <p><%= order.userAddress %></p>
40                 </div>
41             </div>
42             <div class="col-md-6 text-end">
43                 <div class="info-section">
44                     <p><strong>Invoice #: <%= order.orderID %></strong></p>
45                     <p><strong>Date: <%= order.date %></strong></p>
46                 </div>
47             </div>
48         </div>
49         <hr />
50         <!-- End of Invoice Info and Customer Info -->
51
52         <!-- Table -->
53         <table class="table table-hover table-borderless caption-top text-center">
54             <thead>
55                 <tr>
56                     <th scope="col">Product Name</th>
57                     <th scope="col">Quantity</th>
58                     <th scope="col">Unit Price</th>
59                     <th scope="col">Total Price</th>
60                 </tr>
61             </thead>
62             <tbody>
63                 <% products.forEach(product => { %>
64                     <tr>
65                         <td><%= product.productName %></td>
66                         <td><%= product.quantity %></td>
67                         <td>RM<%= product.price.toFixed(2) %></td>
68                         <td>RM<%= product.totalPrice.toFixed(2) %></td>
69                     </tr>
70                     <% }); %>
71                 </tbody>
72             </table>
73             <div class="total">Grand Total: RM<%= grandTotal.toFixed(2) %></div>
74         </div>
75         <!-- End of Table -->
76
77         <!-- Bootstrap JS with Popper -->
78         <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
79     </body>
80 </html>
```

Figure 6. 56 Overview of ‘orderdetails.ejs’

Obviously, the ‘orderdetails.ejs’ will lastly become a template for the PDF.

```

113    <!-- Overlay Message for Invoice Generation -->
114    <div
115      id="loading-overlay"
116      class="position-fixed top-0 start-0 w-100 h-100 bg-dark bg-opacity-50 d-none"
117      style="z-index: 1050"
118    >
119      <div class="d-flex justify-content-center align-items-center w-100 h-100">
120        <div
121          class="alert alert-info text-center"
122          role="alert"
123        >
124          Invoice is generating, please wait...
125        </div>
126      </div>
127    </div>
128  <!-- End of Overlay Message for Invoice Generation -->

```

Figure 6. 57 Overlay Message for Invoice Generation in ‘manageorder.ejs’ and ‘supplierorder.ejs’

Since the generation of a PDF takes a few seconds to process, the overlay message helps inform the user and supplier that the system is functioning normally.

```

221    // Function to show loading overlay
222    function showLoadingOverlay() {
223      const overlay = document.getElementById("loading-overlay");
224      overlay.classList.remove("d-none");
225    }
226
227    //Function to hide the loading overlay
228    function hideLoadingOverlay() {
229      const overlay = document.getElementById("loading-overlay");
230      overlay.classList.add("d-none");
231    }
232
233    // Function to generate invoice to show order details
234    async function generateInvoice(orderID, status) {
235      if (status !== "Confirmed") {
236        try {
237          const response = await fetch(
238            `/orderdetails/unconfirmed/${orderID}`
239          );
240          if (response.ok) {
241            const orderDetails = await response.json();
242            displayOrderDetails(orderDetails);
243            const orderDetailsModal = new bootstrap.Modal(
244              document.getElementById("orderDetailsModal")
245            );
246            orderDetailsModal.show();
247          } else {
248            showMessage("Failed to fetch order details.", "warning");
249          }
250        } catch (error) {
251          showMessage("Error fetching order details.", "warning");
252        }
253      } else {
254        showLoadingOverlay();
255        try {
256          const response = await fetch(`orderdetails/${orderID}`);
257          if (response.ok) {
258            const blob = await response.blob();
259            const url = URL.createObjectURL(blob);
260            window.open(url);
261          } else {
262            showMessage("Failed to generate invoice.", "warning");
263          }
264        } catch (error) {
265          showMessage("Error generating invoice.", "warning");
266        } finally {
267          hideLoadingOverlay();
268        }
269      }
270    }

```

Figure 6. 58 Client-Side JavaScript in ‘manageorder.ejs’

The first two functions manage the behaviour of the overlay message, ensuring it opens and closes as expected. Without these functions, the overlay message would remain open until the webpage is refreshed.

The ‘generateInvoice’ function handles the retrieval and display of order details and generates a PDF invoice. If the order status is not "Confirmed," it fetches unconfirmed order details from the server and displays them in a Bootstrap modal. If the order is "Confirmed," it shows a loading overlay and fetches the order details for PDF generation. The PDF is then opened in a new window. Throughout the process, it displays appropriate messages to inform the user of the system's status and hides the loading overlay once the process is complete.

```

292     // Handles the row click event to show order details or open invoice
293     window.handleRowClick = async function (orderID, orderStatus) {
294       const row = document.querySelector(`tr[onclick*='${orderID}']`);
295       const updatedStatus = row
296         .querySelector("td:nth-child(6) button")
297         .textContent.trim();
298
299       if (
300         updatedStatus === "Pending" ||
301         updatedStatus === "Confirm Order"
302       ) {
303         try {
304           const response = await fetch(
305             `/orderdetails/unconfirmed/${orderID}`
306           );
307           if (response.ok) {
308             const orderDetails = await response.json();
309             displayOrderDetails(orderDetails);
310             const orderDetailsModal = new bootstrap.Modal(
311               document.getElementById("orderDetailsModal")
312             );
313             orderDetailsModal.show();
314           } else {
315             showMessage("Failed to fetch order details.", "warning");
316           }
317         } catch (error) {
318           showMessage("Error fetching order details.", "warning");
319         }
320       } else {
321         // Show the loading overlay
322         showLoadingOverlay();
323
324         // Open the invoice in a new window
325         setTimeout(() => {
326           window.open(`/orderdetails/${orderID}`, "_blank");
327           hideLoadingOverlay(); // Hide overlay in the original window
328         }, 1000); // Simulate a delay for the overlay to be visible
329       }
330     };

```

Figure 6. 59 Client-Side JavaScript in ‘supplierorder.ejs’

The client-side JavaScript in ‘supplierorder.ejs’ also has the mechanism to display PDF invoice, manage overlay message behaviour. The key difference is that it does not use blobs to generate the invoice; instead, it directly opens the URL of the generated PDF in a new window, providing a seamless user experience for viewing the invoice.

6.8.1.3 F011 Check Order Details

The implementation of this functionality aligns with real-world business logic, ensuring that an invoice is created only when both parties agree to proceed with the transaction. This functionality also provides the user and supplier an ability to check the order before order confirmation. It shares the same model functions, ‘getOrderDetailsById’ and ‘getOrderProducts’, with the previous functionality.

6.8.1.3.1 Route

```
86  /**
87  * @route GET /orderdetails/unconfirmed/:orderID
88  * @desc Show unconfirmed details of a specific order
89  * @access Public
90  */
91 router.get('/orderdetails/unconfirmed/:orderID', mainController.showUnconfirmedOrderDetails);
```

Figure 6. 60 Route for Unconfirmed Order Detail Rendering in ‘main.js’

This route is used to deal with the display of unconfirmed order detail for user and supplier.

6.8.1.3.2 Controller

```
298 /**
299 * @desc Renders the order details of an unconfirmed order from the database and returns them as a JSON response
300 * @param {Object} req - Express request object
301 * @param {Object} req.params - Parameters from the request URL
302 * @param {number} req.params.orderID - The ID of the order to be displayed
303 * @param {Object} res - Express response object
304 * @returns {Promise<void>}
305 * @throws {Error} - Throws an error if there is an issue fetching the order details or products
306 */
307 exports.showUnconfirmedOrderDetails = async (req, res) => {
308   try {
309     const orderID = req.params.orderID;// Get the order ID from the request parameters
310     const orderDetails = await mainModels.getOrderDetailsById(orderID);
311     const products = await mainModels.getOrderProducts(orderID)
312
313     orderDetails.products = products;
314
315     res.json(orderDetails);
316   } catch (error) {
317     console.error('Failed to fetch order details:', error);
318     res.status(500).json({ message: 'Failed to load order details.' });
319   }
320};
```

Figure 6. 61 Controller for Displaying Unconfirmed Order Details in ‘mainController.js’

This code is designed to fetch and render the details of an unconfirmed order from the database. It combines these details and products into a single object and sends them as a JSON response. It means that this functionality is achieved with the corporation of server and client-side.

6.8.1.3.3 View

```
205      <!-- Modal for Unconfirmed Orders -->
206      <div
207          | class="modal fade"
208          | id="orderDetailsModal"
209          | tabindex="-1"
210          | aria-labelledby="orderDetailsModalLabel"
211          | aria-hidden="true"
212      >
213      <div class="modal-dialog modal-lg modal-dialog-centered">
214          <div class="modal-content">
215              <div class="modal-header">
216                  <h5
217                      | class="modal-title"
218                      | id="orderDetailsModalLabel"
219                  >
220                      Order Details
221                  </h5>
222                  <button
223                      | type="button"
224                      | class="btn-close"
225                      | data-bs-dismiss="modal"
226                      | aria-label="Close"
227                  ></button>
228              </div>
229              <div class="modal-body">
230                  <div id="orderDetailsContent">
231                      | <!-- Order details will be injected here by JavaScript -->
232                  </div>
233              </div>
234              <div class="modal-footer">
235                  <button
236                      | type="button"
237                      | class="btn btn-secondary"
238                      | data-bs-dismiss="modal"
239                  >
240                      Close
241                  </button>
242              </div>
243          </div>
244      </div>
245      </div>
246      <!-- End of Modal for Unconfirmed Orders -->
```

Figure 6. 62 Unconfirmed Order Modal in ‘manageorder.ejs’

This code snippet demonstrates how the modal which is used for displaying the detail information of unconfirmed order is set up. The order details will then be injected in line 231 by using client-side JavaScript in ‘manageorder.ejs’ and ‘supplierorder.ejs’ respectively.

```

272 // Function to display order details in modal
273 function displayOrderDetails(orderDetails) {
274   const orderDetailsContent = document.getElementById(
275     "orderDetailsContent"
276   );
277   orderDetailsContent.innerHTML =
278     `

Supplier Name: ${orderDetails.supplierCompanyName}


279     </p>
280     <p>Supplier Address: ${orderDetails.supplierAddress}
281     </p>
282     <p>Delivery Date: ${orderDetails.deliveryDate}</p>
283     <p>Products:</p>
284     <table class="table table-hover table-borderless caption-top text-center">
285       <thead>
286         <tr>
287           <th>Product Name</th>
288           <th>Quantity</th>
289           <th>Price</th>
290           <th>Total</th>
291         </tr>
292       </thead>
293       <tbody>
294         ${orderDetails.products
295           .map(
296             (product) => `
297               <tr>
298                 <td>${product.productName}</td>
299                 <td>${product.quantity}</td>
300                 <td>${product.price}</td>
301                 <td>${product.totalPrice.toFixed(2)}</td>
302               </tr>
303             )
304           .join("")
305         )
306       </tbody>
307     </table>
308     <p>Grand Total: ${orderDetails.totalPrice.toFixed(
309       2
310     )}</p>
311   `;
312 }
313 
```

Figure 6. 63 Client-Side JavaScript in ‘userorder.ejs’

```

423 // Displays order detail in modal
424 function displayOrderDetails(orderDetails) {
425   let grandTotal = 0;
426   const orderDetailsContent = document.getElementById(
427     "orderDetailsContent"
428   );
429   orderDetailsContent.innerHTML =
430     `

CompanyName: ${orderDetails.userCompanyName}</p>
431     <p>Company Address: ${orderDetails.userAddress}</p>
432     <p>Delivery Date: ${orderDetails.deliveryDate}</p>
433     <p>Products:</p>
434     <table class="table table-hover table-borderless caption-top text-center">
435       <thead>
436         <tr>
437           <th>Product Name</th>
438           <th>Quantity</th>
439           <th>Price</th>
440           <th>Total</th>
441         </tr>
442       </thead>
443       <tbody>
444         ${orderDetails.products
445           .map(
446             (product) => `
447               <tr>
448                 <td>${product.productName}</td>
449                 <td>${product.quantity}</td>
450                 <td>${product.price}</td>
451                 <td>${product.totalPrice.toFixed(2)}</td>
452               </tr>
453             )
454           .join("")
455         )
456       </tbody>
457     </table>
458     <p>Grand Total: ${orderDetails.totalPrice.toFixed(
459       2
460     )}</p>
461   `;
462 }
463


```

Figure 6. 64 Client-Side JavaScript in ‘supplierorder.ejs’

Both figures indicate the required information that needs to be displayed in unconfirmed order detail modal is set up.

6.8.1.4 F013 Upload Profile Photo

6.8.1.4.1 Route

```
70  /**
71   * @route POST /user/uploadProfilePhoto
72   * @desc Handle profile photo upload
73   * @access Private
74   */
75  router.post('/user/uploadProfilePhoto', isAuthenticated, mainController.uploadProfilePhoto);
```

Figure 6. 65 Route for Upload Profile Photo Request Handling in ‘main.js’

This route handles POST requests to ‘/user/uploadProfilePhoto’. When a user submits the upload-profile-photo form, this route triggers the ‘uploadProfilePhoto’ function in ‘mainController.js’ to process the request.

6.8.1.4.2 Model

```
130  /**
131   * @desc Updates the profile photo path for a specific user in the database
132   * @param {number} userId - The ID of the user whose profile photo is being updated
133   * @param {string} imagePath - The new path to the user's profile photo
134   * @returns {Promise<number>} - A promise that resolves with the number of rows affected
135   * @throws {Error} - Throws an error if there is an issue with the database query
136   */
137  updateProfilePhoto: function(userId, imagePath) {
138    return new Promise((resolve, reject) => {
139      console.log(`Updating user ${userId} with new image path ${imagePath}`);
140      const query = 'UPDATE Users SET profilePhoto = ? WHERE userID = ?';
141      db.run(query, [imagePath, userId], function(err) {
142        if (err) {
143          console.error("Database update error:", err);
144          return reject(err);
145        }
146        console.log(`New profile Photo updated successfully`);
147        resolve(this.changes);
148      });
149    });
150  },
```

Figure 6. 66 Model for Uploading Profile Photo in ‘mainModels.js’

This code snippet is a model function in mainModels.js designed to update the profile photo path for a specific user in the database. It uses a promise to handle asynchronous database operations, logging both successes and errors for debugging purposes.

To ensure the profile photo is inserted or updated correctly, the SQL query is prepared to update the ‘profilePhoto’ field for user with the specified ‘userId’.

6.8.1.4.3 Controller

```
9  /**
10   * @desc Configures storage settings for Multer
11   * - Sets destination folder for profile photos
12   * - Sets a unique filename using the current timestamp
13   */
14 const storage = multer.diskStorage({
15   destination: './public/uploads/profilephoto/',
16   filename: function(req, file, cb) {
17     cb(null, 'profile-' + Date.now() + path.extname(file.originalname));
18   }
19 });
20 /**
21  * @desc Configures Multer middleware for file upload
22  * - Uses the specified storage configuration
23  * - Filters files to allow only JPEG, JPG, and PNG formats
24  */
25 const upload = multer({
26   storage: storage,
27   limits: { fileSize: 1000000000 },
28   fileFilter: function(req, file, cb) {
29     checkFileType(file, cb);
30   }
31 }).single('profilePhoto');
32 /**
33  * @desc Checks the file type to ensure it is a valid image format (JPEG, JPG, PNG)
34  * @param {Object} file - The file to be uploaded
35  * @param {Function} cb - Callback function to indicate if the file type is valid
36  */
37 function checkFileType(file, cb) {
38   const filetypes = /jpeg|jpg|png/;
39   const extname = filetypes.test(path.extname(file.originalname).toLowerCase());
40   const mimetype = filetypes.test(file.mimetype);
41   if (mimetype && extname) {
42     return cb(null, true);
43   } else {
44     cb('Error: Only JPEG, JPG, and PNG files are allowed!');
45   }
46 }
47 }
48 }
```

Figure 6. 67 Setting Up Mutler for Profile Photo Upload in ‘mainController.js’

It first sets up the Mutler middleware for handling file uploads, especially for profile photos. Then, the uploaded files will be stored to ‘./public/uploads/profilephoto/’. The filename will then be set with an unique filename.

Besides that, the function, in line26-32, is used to set up the Mutler Middleware. It sets the storage, limit storage of file, ‘checkFileType’ , and specifies that only a single file named will be uploaded.

The ‘checkFileType’ is a mechanism to ensure that only the files with valid image formats such as . JPEG, JPG, and PNG are allowed to choose.

```

188  /**
189   * @desc Handles the profile photo upload for a user
190   * @param {Object} req - Express request object
191   * @param {Object} req.session - Session object containing user session data
192   * @param {Object} req.session.user - The current logged-in user's session data
193   * @param {Object} res - Express response object
194   * @throws {Error} - Throws an error if there is an issue during the signup process
195   */
196  exports.uploadProfilePhoto = (req, res) => {
197    upload(req, res, (err) => {
198      if (err) {
199        console.error("Upload Error:", err);
200        return res.status(400).json({ success: false, message: 'Upload failed: ' + err });
201      }
202      if (!req.file) {
203        return res.status(400).json({ success: false, message: 'No file selected!' });
204      }
205      const imagePath = `/uploads/profilephoto/${req.file.filename}`;
206      mainModels.updateProfilePhoto(req.session.user.id, imagePath)
207        .then(() => {
208          // Update the session with the new image path
209          req.session.user.profilePhoto = imagePath;
210          // Save the session before responding
211          req.session.save(err => {
212            if (err) {
213              console.error('Session save error:', err);
214              return res.status(500).json({ success: false, message: 'Session save error.' });
215            }
216            res.json({ success: true, imagePath: imagePath });
217          });
218        })
219        .catch(err => {
220          console.error("Database update failed:", err);
221          res.status(500).json({ success: false, message: 'Database update failed: ' + err.message });
222        });
223    });
224  };

```

Figure 6. 68 Controller for Uploading Profile Photo in ‘mainController.js’

This code snippet is from the ‘mainController.js’ file and is responsible for handling the upload of profile photos for users. The process involves several steps, including file upload, database update, and session management.

The file upload is handled using the ‘upload’ function, previously configured with Multer. Once the file is successfully uploaded, the function constructs the ‘imagePath’ using the uploaded file’s filename and calls the ‘updateProfilePhoto’ function from ‘mainModels’ to update the user’s profile photo path in the database. The function passes the user ID from the session and the new ‘imagePath’.

After updating the database, it updates the user’s session with the new ‘imagePath’ and saves the session to ensure the changes are persistent in system interface.

6.8.1.4.4 View

```
1  <!-- Profile Photo Upload Modal -->
2  <div
3  |   class="modal fade"
4  |   id="uploadModal"
5  |   tabindex="-1"
6  |   aria-labelledby="uploadModalLabel"
7  |   aria-hidden="true"
8  >
9  <div class="modal-dialog modal-dialog-centered">
10 <div class="modal-content">
11 <div class="modal-header">
12 <h5
13 |   class="modal-title"
14 |   id="uploadModalLabel"
15 >
16   Upload Profile Photo
17 </h5>
18 <button
19 |   type="button"
20 |   class="btn-close"
21 |   data-bs-dismiss="modal"
22 |   aria-label="Close"
23 ></button>
24 </div>
25 <div class="modal-body">
26 <form
27 |   id="uploadForm"
28 |   enctype="multipart/form-data"
29 >
30 <div class="mb-3">
31 <label
32 |   for="profilePhoto"
33 |   class="form-label"
34 >Choose a photo (PNG, JPG):</label>
35 >
36 <input
37 |   type="file"
38 |   class="form-control"
39 |   id="profilePhoto"
40 |   name="profilePhoto"
41 |   accept=".png, .jpg, .jpeg"
42 |   required
43 >/>
44 </div>
45 <div class="modal-footer">
46 <button
47 |   type="button"
48 |   class="btn btn-secondary"
49 |   data-bs-dismiss="modal"
50 >
51   Close
52 </button>
53 <button
54 |   type="submit"
55 |   class="btn btn-primary"
56 >
57   Upload
58 </button>
59 </div>
60 </form>
61 </div>
62 </div>
63 </div>
64 </div>
65 <!-- End of Profile Photo Upload Modal -->
```

Figure 6. 69 Profile Photo Upload Modal in ‘uploadprofilemodal.ejs’

Using the Bootstrap framework to manage this modal, user and supplier need to upload their profile photos through this interface. This partial is also included in the profile options modal of user and supplier respectively. It requires client-side JavaScript to manage the POST request to route and interaction stuffs.

```

67   <!-- Client Side JS -->
68 <script>
69   document
70     .getElementById("uploadForm")
71     .addEventListener("submit", function (event) {
72       event.preventDefault();
73
74     const formData = new FormData(this);
75
76     fetch("/user/uploadProfilePhoto", {
77       method: "POST",
78       body: formData,
79     })
80       .then((response) => response.json())
81       .then((data) => {
82         if (data.success) {
83           console.log("Upload successful:", data.imagePath);
84           // Update the profile photo in the UI
85           const profilePhotoDisplayNavbar =
86             document.querySelector(".nav-link img");
87           const profilePhotoDisplayProfile =
88             document.getElementById("profile-photo");
89
90           if (profilePhotoDisplayNavbar) {
91             profilePhotoDisplayNavbar.src =
92               data.imagePath + "?" + new Date().getTime(); // Add a timestamp to avoid caching issues
93             console.log(
94               "Profile photo updated in navbar to:",
95               data.imagePath
96             );
97           } else {
98             console.error(
99               "Profile photo display element in navbar not found."
100             );
101           }
102
103           if (profilePhotoDisplayProfile) {
104             profilePhotoDisplayProfile.src =
105               data.imagePath + "?" + new Date().getTime(); // Add a timestamp to avoid caching issues
106             console.log(
107               "Profile photo updated in profile info to:",
108               data.imagePath
109             );
110           } else {
111             console.error(
112               "Profile photo display element in profile info not found."
113             );
114           }
115           // Close the modal
116           const uploadModal = bootstrap.Modal.getInstance(
117             document.getElementById("uploadModal")
118           );
119           if (uploadModal) {
120             uploadModal.hide();
121           } else {
122             console.error("Upload modal instance not found.");
123           }
124           } else {
125             alert(data.message);
126           }
127         }
128       .catch((error) => {
129         console.error("Error:", error);
130         alert("An error occurred while uploading the profile photo.");
131       });
132     });
133   </script>
134 <!-- End of Client Side JS -->

```

Figure 6. 70 Client-Side JavaScript 'uploadprofilemodal.ejs'

This script handles the client-side logic for uploading a profile photo, including form submission, server communication, updating the UI with the new photo, and error handling. It ensures a smooth user experience by providing feedback and managing the upload process effectively. For instance, the profile photo will be displayed on navigation bar and profile options modal. This code helps to ensure that the demonstration of profile photo can be changed immediately without being refreshed on the webpage.

6.8.1.5 F015 Search Order

6.8.1.5.1 Route

```
70  /**
71  * @route GET /searchOrders
72  * @desc Search for orders by customer name
73  * @access Private (User only)
74  */
75 router.get('/searchOrders', isAuthenticated, userController.searchOrders);
```

Figure 6. 71 Route for User Manage Order Page Rendering in ‘userRoutes.js’

```
100 /**
101 * @route POST /searchOrders
102 * @desc Search orders by customer name
103 * @access Private (Supplier only)
104 */
105 router.post('/searchOrders', isAuthenticated, supplierController.searchOrders);
```

Figure 6. 72 Route for Supplier Manage Order Page Rendering in ‘supplierRoutes.js’

Both code snippets in ‘userRoutes.js’ and ‘supplierRoutes.js’ serve the same purpose, which is dynamically rendering the searched orders based on the searched ‘company name’. However, the only difference is that each of them deals with the ‘GET’ and ‘POST’ request respectively.

The routes for supplier would be ‘<http://localhost:3000/supplier/searchOrders>’. On the other hand, the route for user is ‘<http://localhost:3000/user/searchOrders?searchQuery=>’.

6.8.1.5.2 Model

```
232 /**
233 * @desc Searches for orders by supplier name for a specific user
234 * @param {number} userID - The ID of the user whose orders are being searched
235 * @param {string} searchQuery - The search query used to find orders by supplier name
236 * @returns {Promise<Array>} - A promise that resolves with an array of orders matching the search query
237 * @throws {Error} - Throws an error if there is an issue with the database query
238 */
239 searchOrdersBySupplierName: function(userID, searchQuery) {
240     return new Promise((resolve, reject) => {
241         // SQL query to find orders by supplier name for a specific user
242         const query = `
243             SELECT
244                 Orders.orderID,
245                 Orders.userID,
246                 Users.companyName AS supplierName,
247                 Orders.date,
248                 Orders.deliveryDate,
249                 Orders.totalPrice,
250                 Orders.deliveryStatus AS status
251             FROM Orders
252             INNER JOIN Suppliers ON Orders.supplierID = Suppliers.supplierID
253             INNER JOIN Users ON Suppliers.userID = Users.userID
254             WHERE Orders.userID = ? AND Users.companyName LIKE ?
255         `;
256         db.all(query, [userID, `%${searchQuery}%`], (err, rows) => {
257             if (err) {
258                 console.error('Error searching orders:', err);
259                 reject(err);
260             } else {
261                 resolve(rows);
262             }
263         });
264     });
265 },
266
267 };
```

Figure 6. 73 Model for Searching Order in ‘userModels.js’

This function, ‘searchOrdersBySupplierName’, is used to retrieve the ‘orderID’, ‘userID’, ‘supplierName’, ‘date’, ‘deliveryDate’, ‘totalPrice’, ‘deliveryStatus’ based on the searched supplier name. The ‘searchQuery’ parameter is the data which the user enters in the navigation bar to find orders associated with a specific supplier.

6.8.1.5.3 Controller

```

237 /**
238 * @desc Searches for orders by supplier name and renders the manage orders page with the search results
239 * @param {Object} req - Express request object
240 * @param {Object} req.session - Session object containing user session data
241 * @param {Object} req.session.user - The current logged-in user's session data
242 * @param {number} req.session.user.id - The ID of the current user
243 * @param {Object} req.query - Query parameters from the request URL
244 * @param {string} req.query.searchQuery - The search query used to find orders by supplier name
245 * @param {Object} res - Express response object
246 * @returns {Promise<void>}
247 * @throws {Error} - Throws an error if there is an issue searching for orders
248 */
249 exports.searchOrders = async (req, res) => {
250   const { searchQuery } = req.query;
251   const userID = req.session.user.id;
252
253   try {
254     const orders = await userModels.searchOrdersBySupplierName(userID, searchQuery);
255
256     console.log('All searched orders retrieved successfully.');
257     res.render('user/manageorder', {
258       user: req.session.user,
259       orders
260     });
261   } catch (error) {
262     console.error('Failed to search orders:', error);
263     res.status(500).render('error', { message: 'Failed to search orders.' });
264   }
265 };

```

Figure 6. 74 Controller for Searching Order in ‘userController.js’

This code snippet is a controller function for searching orders by supplier name in ‘userController.js’. When the function is invoked, it extracts the ‘searchQuery’ from the request query parameters and the ‘userID’ from the user session data. The function then attempts to search for orders by calling the ‘searchOrdersBySupplierName’ method from ‘userModels’, passing the ‘userID’ and ‘searchQuery’ as arguments. If the search is successful, the retrieved orders are logged with a success message and rendered to the 'user/manageorder' view along with the user session data.

```

279 /**
280 * @desc Searches for orders by customer name for a specific supplier and renders the supplier orders page with the search results
281 * @param {Object} req - Express request object
282 * @param {Object} req.session - Session object containing user session data
283 * @param {Object} req.session.user - The current logged-in user's session data
284 * @param {number} req.session.user.supplierID - The ID of the supplier
285 * @param {Object} req.body - Request body containing the search query
286 * @param {string} req.body.searchQuery - The search query used to find orders by customer name
287 * @param {Object} res - Express response object
288 * @returns {Promise<void>}
289 * @throws {Error} - Throws an error if there is an issue searching for orders
290 */
291 exports.searchOrders = async (req, res) => {
292   try {
293     const supplierID = req.session.user.supplierID;
294     const { searchQuery } = req.body;
295     const orders = await supplierModels.getOrdersByCustomerName(supplierID, searchQuery);
296     const description = await supplierModels.getSupplierDescription(supplierID);
297
298     console.log('All searched orders retrieved successfully.');
299     res.render('supplier/supplierorder', {
300       user: req.session.user,
301       orders,
302       searchQuery, // Pass the search query back to the template
303       description: description || '' // Pass the description to the template
304     });
305   } catch (error) {
306     console.error('Failed to search orders:', error);
307     res.status(500).send('Failed to search orders. Error: ' + error.message);
308   }
309 };

```

Figure 6. 75 Controller for Searching Order in ‘supplierController.js’

This code snippet is mostly the same as the code in ‘userController.js’. However, it retrieves the supplier’s description using the ‘getSupplierDescription’ method. This is used to prevent the error when system execution.

6.8.1.5.4 View

```

30 <!-- Header, Search Bar, and Orders Table Section -->
31 <div class="container">
32   <!-- Header and Search Bar -->
33   <h1 class="header sticky-top">Manage Orders</h1>
34   <!-- Search Order Bar -->
35   <div class="search-bar input-group mb-3 sticky-top">
36     <input
37       type="text"
38       id="searchOrderInput"
39       class="form-control"
40       placeholder="Search order by supplier name"
41       aria-label="Search order"
42       aria-describedby="button-addon2"
43       onkeydown="if (event.key === 'Enter') searchOrders()"
44     />
45     <button
46       class="btn btn-outline-secondary"
47       type="button"
48       id="button-addon2"
49       onclick="searchOrders()"
50     >
51       | Search
52     </button>
53   </div>
54   <!-- End of Search Order Bar -->
55   <!-- End of Header and Search Bar -->

```

Figure 6. 76 Search Order Section in ‘manageorder.ejs’

```

36  |    <!-- Header, Search Bar, and Orders Table Section -->
37  |    <div class="container">
38  |        <!-- Header and Search Bar -->
39  |        <h1 class="header sticky-top">Manage Orders</h1>
40  |        <!-- Search Order Bar -->
41  |        <div class="search-bar input-group mb-3 sticky-top">
42  |            <form
43  |                action="/supplier/searchOrders"
44  |                method="POST"
45  |                class="d-flex w-100"
46  |            >
47  |                <input
48  |                    type="text"
49  |                    name="searchQuery"
50  |                    class="form-control"
51  |                    placeholder="Search order"
52  |                    aria-label="Search order"
53  |                    aria-describedby="button-addon2"
54  |                    value="<%= searchQuery %>""
55  |                />
56  |                <button
57  |                    class="btn btn-outline-secondary"
58  |                    type="submit"
59  |                    id="button-addon2"
60  |                >
61  |                    Search
62  |                </button>
63  |            </form>
64  |        </div>
65  |    <!-- End of Search Order Bar -->

```

Figure 6. 77 Search Order Section in ‘supplierorder.ejs’

Both figures represent the search order bar in each ‘manageorder.ejs’ and ‘supplierorder.ejs’. As demonstrated code, each uses different techniques. The ‘manageorder.ejs’ requires a client-side script to complement the function.

```

316  |    // Function to search order
317  |    function searchOrders() {
318  |        const searchQuery = document.getElementById("searchOrderInput").value;
319  |        window.location.href = `/user/searchOrders?searchQuery=${encodeURIComponent(
320  |            searchQuery
321  |        )}`;
322  |    }

```

Figure 6. 78 Client-Side JavaScript in ‘userorder.ejs’

This section contains an HTML form used for searching orders. The form uses the POST method to send the search query to the ‘/user/searchOrders’ route. This setup allows users to search orders by typing in a search query and either submitting the form or pressing ‘Enter’, which triggers the client-side JavaScript function to handle the redirection and display the search results.

6.8.1.6 F017 Sign Out

6.8.1.6.1 Route

```
53 /**
54  * @route POST /logout
55  * @desc Log out the user by destroying the session and clearing the cookie
56  * @access Private
57 */
58 router.post('/logout', isAuthenticated, (req, res) => {
59   req.session.destroy(err => {
60     if (err) {
61       console.error('Failed to destroy the session during logout.', err);
62       return res.status(500).send("Could not log out, please try again.");
63     }
64     res.clearCookie('connect.sid');
65     res.redirect('/login');
66   });
67 });


```

Figure 6. 79 Route for Sign Out Request Handling in ‘main.js’

This route verifies the user is authenticated using the ‘isAuthenticated’ middleware. Then, the current user session is destroyed when the request is received. To ensure that the user is fully logged out, it also clears the session with the ‘clearCookie’. Lastly, it will redirect the user to the login page, this process is deemed as the completion of sign-out process.

6.8.1.6.2 View

```
1  <!-- Sign Out Confirmation Modal -->
2  <div
3    class="modal fade"
4    id="signOutConfirmationModal"
5    tabindex="-1"
6    aria-labelledby="signOutConfirmationModalLabel"
7    aria-hidden="true"
8  >
9    <div class="modal-dialog modal-dialog-centered">
10      <div class="modal-content">
11        <div class="modal-header">
12          <h5
13            class="modal-title"
14            id="signOutConfirmationModalLabel"
15          >
16            Confirm Sign Out
17          </h5>
18          <button
19            type="button"
20            class="btn-close"
21            data-bs-dismiss="modal"
22            aria-label="Close"
23          ></button>
24        </div>
25        <div class="modal-body">
26          <p>Are you sure you want to sign out?</p>
27        </div>
28        <div class="modal-footer">
29          <button
30            type="button"
31            class="btn btn-secondary"
32            data-bs-dismiss="modal"
33          >
34            Cancel
35          </button>
36          <!-- Form for logging out -->
37          <form
38            action="/logout"
39            method="POST"
40            style="display: inline">
41            <button
42              type="submit"
43              class="btn btn-danger"
44            >
45              | Sign Out
46            </button>
47          </form>
48        </div>
49      </div>
50    </div>
51  </div>
52 </div>
53 <!-- End of Sign Out Confirmation Modal -->
```

Figure 6. 80 Sign Out Confirmation Modal in ‘signoutmodal.ejs’

This modal will be triggered when the ‘Sign Out’ button is clicked on profile options modal. Based on this code, when the ‘Sign Out’ button is clicked, the request will be posted to ‘/logout’ by using POST method.

```
55  <!-- Client Side JS -->
56  <script>
57  | document.addEventListener("DOMContentLoaded", () => {
58  |   const logoutForm = document.querySelector('form[action="/logout"]');
59  |   if (logoutForm) {
60  |     logoutForm.addEventListener("submit", () => {
61  |       // sessionStorage.clear(); // Clear the sessionStorage
62  |     });
63  |   }
64  | });
65  </script>
66  <!-- End of Client Side JS -->
```

Figure 6. 81 Client-Side JavaScript in ‘signoutmodal.ejs’

The client-side JavaScript in ‘signoutmodal.ejs’ ensures that when the logout form is submitted, it can perform additional actions, such as clearing the session storage. The event listener waits until the DOM is fully loaded before attaching the ‘submit’ event listener to the logout form. Basically, the DOM allows web pages to be dynamic and interactive.

6.8.2 User Functionality

6.8.2.1 F003 Select Supplier

6.8.2.1.1 Route

```
14  /**
15  * @route GET /userhome
16  * @desc Show user home page
17  * @access Private (User only)
18  */
19 router.get('/userhome', isAuthenticated, (req, res, next) => {
20   if (!req.session.user || req.session.user.type !== 'User') {
21     return res.redirect('/login');
22   }
23   // If the session user is of type 'User', continue to the next middleware, which is the userController's showUserHome function
24   next(); // Pass control to the next middleware function, which is userController.showSupplierHome
25 }, userController.showUserHome);
```

Figure 6. 82 Route for User Homepage Rendering in ‘userRoutes.js’

This is a code snippet in ‘userRoutes.js’ file. It helps to show the user homepage for selecting supplier with the URL ‘<http://localhost:3000/user/userhome>’. As there are a few accessibility roles in WoodChain, this route also provides a mechanism for ensuring that the session user exists, and their type is ‘User’. So, only the ‘User’ can access the user homepage.

6.8.2.1.2 Model

```
7  /**
8  * @desc Retrieves a list of all suppliers with their associated user details
9  * @returns {Promise<Array>} - A promise that resolves with an array of supplier details
10 * @throws {Error} - Throws an error if there is an issue with the database query
11 */
12 getAllSuppliers: function() {
13   return new Promise((resolve, reject) => {
14     // SQL query to fetch supplier details along with their user information
15     const query = `
16       SELECT Users.userID, Users.companyName, Users.companyAddress, Users.profilePhoto, Suppliers.supplierID, Suppliers.supplierDescription
17       FROM Users
18       JOIN Suppliers ON Users.userID = Suppliers.userID
19     `;
20     db.all(query, [], (err, rows) => {
21       if (err) {
22         console.error('Error fetching suppliers:', err);
23         reject(err);
24       } else {
25         resolve(rows);
26       }
27     });
28   },
29 }
```

Figure 6. 83 Model for Retrieving Supplier Information in ‘userModels.js’

The ‘getAllSuppliers’ function is used to retrieve all the suppliers that have been recorded in the database, allowing users to choose their preferred supplier for product selection. In short, it is important for allowing users to view and select suppliers for their needs.

The function constructs an SQL query that joins the Users and Suppliers tables. It selects user-specific fields (‘userID’, ‘companyName’, ‘companyAddress’, ‘profilePhoto’) and supplier-specific fields (‘supplierID’, ‘supplierDescription’). It ensures the interface will be able to fully render the information.

6.8.2.1.3 Controller

```
6  /**
7  * @desc Renders the user home page with a list of all suppliers
8  * @param {Object} req - Express request object
9  * @param {Object} req.session - Session object containing user session data
10 * @param {Object} req.session.user - The current logged-in user's session data
11 * @param {Object} res - Express response object
12 * @returns {Promise<void>}
13 * @throws {Error} - Throws an error if there is an issue fetching suppliers
14 */
15 exports.showUserHome = async (req, res) => {
16   try {
17     const suppliers = await userModel.getAllSuppliers();
18
19     // console.log('Suppliers:', suppliers); // Debug output to confirm data fetching
20     console.log('All suppliers retrieved successfully.');
21     res.render('user/userhome', {
22       user: req.session.user,
23       suppliers,
24       searchQuery: '' //Add default empty search query
25     });
26   } catch (error) {
27     console.error('Failed to fetch suppliers:', error);
28     res.status(500).render('error', { message: 'Failed to load suppliers.' });
29   }
30};
```

Figure 6. 84 Controller for Displaying User Homepage in 'userController.js'

This code defines a controller function named 'showUserHome' in 'userRoutes.js', which is responsible for rendering the user homepage with a list of all suppliers. When a user accesses this route, the function begins by attempting to fetch all suppliers from the database using 'userModel.getAllSuppliers()'. This asynchronous call ensures efficient data retrieval without blocking the execution thread. Upon successfully retrieving the supplier data, the function logs a confirmation message and renders the userhome view, passing the user's session data, the list of suppliers, and an empty search query to the template. This allows the user to view and interact with the supplier information on their homepage.

6.8.2.1.4 View

```
30      <!-- Header, Search Bar, and Supplier List Section -->
31  <div class="container">
32    <!-- Header and Search Bar -->
33    <h1 class="header sticky-top">Supplier List</h1>
34    <!-- Search Supplier Bar -->
35    <div class="search-bar input-group mb-3 sticky-top">...
36    </div>
37    <!-- End of Search Supplier Bar -->
38
39    <!-- Supplier List -->
40    <ul class="list-group">
41      <% if (suppliers && suppliers.length > 0) { %> <%
42        suppliers.forEach(supplier => { %>
43          <li class="list-group-item d-flex align-items-center">
44            <div class="col-auto me-3">
45              <img
46                src=<%= supplier.profilePhoto ? supplier.profilePhoto : '/images/usericon.png' %>
47                alt="Supplier Icon"
48                class="img-fluid rounded-circle"
49                style="width: 32px; height: 32px"
50              />
51            </div>
52            <div class="col">
53              <h5 class="mb-0"><%= supplier.companyName %></h5>
54              <p class="mb-0 text-muted">
55                <%= supplier.supplierDescription.length > 70 ?
56                  supplier.supplierDescription.substring(0, 70) + '...' :
57                  supplier.supplierDescription %>
58              </p>
59            </div>
60            <a
61              href="/user/selectproduct/<%= supplier.supplierID %>"
62              class="stretched-link"
63            ></a>
64          </li>
65        <% }); %> <% } else { %>
66        <li class="list-group-item">No suppliers found.</li>
67      <% } %>
68    </ul>
69    <!-- End of Supplier List -->
70  </div>
71  <!-- End of Header, Search Bar, and Supplier List Section -->
72
73  <!-- Partial Modals -->
74  <%- include('../partials/usermodals') %> <%
75  include('../partials/signoutmodal') %> <%
76  include('../partials/uploadprofilephotomodal') %>
77  <!-- End of Partial Modals -->
```

Figure 6. 85 Supplier List Section in 'userhome.ejs'

This code represents the syntax for dynamically rendering the supplier information.

6.8.2.2 F004 Select Product

6.8.2.2.1 Route

```
36  /**
37  * @route GET /selectproduct/:supplierID?
38  * @desc Show select product page with an optional supplier ID
39  * @access Private (User only)
40  */
41 router.get('/selectproduct/:supplierID?', isAuthenticated, (req, res, next) => {
42   next();
43 }, userController.showSelectProduct);
```

Figure 6. 86 Route for Select Product Page Rendering in ‘userRoutes.js’

This code snippet defines a route for rendering the select product page in ‘userRoutes.js’. The route is defined as a GET request to ‘/selectproduct/:supplierID’, where ‘supplierID’ is an optional parameter.

6.8.2.2.2 Model

```
59 /**
60  * @desc Retrieves a supplier's details by supplier ID
61  * @param {number} supplierId - The ID of the supplier to retrieve
62  * @returns {Promise<Object|null>} - A promise that resolves with the supplier's details or null if no supplier is found
63  * @throws {Error} - Throws an error if there is an issue with the database query
64 */
65 getSupplierById: function(supplierId) {
66   return new Promise((resolve, reject) => {
67     // SQL query to retrieve supplier details by supplier ID
68     const query = `SELECT Users.userID, Users.companyName, Users.companyAddress, Users.profilePhoto, Suppliers.supplierID, Suppliers.supplierDescription
69       FROM Users
70       JOIN Suppliers ON Users.userID = Suppliers.userID
71       WHERE Suppliers.supplierID = ?`;
72     db.get(query, [supplierId], (err, row) => {
73       if (err) {
74         console.error("Error executing query:", err);
75         reject(err);
76       } else {
77         if (row) {
78           resolve(row);
79         } else {
80           console.log("No supplier found with ID:", supplierId);
81           resolve(null); // or reject(new Error("No supplier found"));
82         }
83       }
84     });
85   },
86 }
```

Figure 6. 87 Model for Retrieving Supplier Details in ‘userModels.js’

This defines a function ‘getSupplierById’ that retrieves the details of a supplier based on their ‘supplierID’. The function constructs a SQL query to join the ‘Users’ and ‘Suppliers’ tables and select relevant details such as ‘userID’, ‘companyName’, ‘companyAddress’, ‘profilePhoto’, ‘supplierID’, and ‘supplierDescription’. It then executes the query using the provided ‘supplierID’ parameter.

```

89  /**
90  * @desc Retrieves all products for a specific supplier by supplier ID
91  * @param {number} supplierId - The ID of the supplier whose products are being retrieved
92  * @returns {Promise<Array>} - A promise that resolves with an array of products
93  * @throws {Error} - Throws an error if there is an issue with the database query
94  */
95  getProductsBySupplierId: function(supplierId) {
96      return new Promise((resolve, reject) => {
97          // SQL query to retrieve products by supplier ID
98          const query = `SELECT * FROM Products WHERE supplierID = ? AND isActive = 1`;
99          db.all(query, [supplierId], (err, rows) => {
100              if (err) {
101                  reject(err);
102              } else {
103                  resolve(rows);
104              }
105          });
106      },
107  );

```

Figure 6. 88 Model for Retrieving Products by Supplier ID in ‘userModels.js’

This defines a function ‘getProductsBySupplierId’ that retrieves all products associated with a specific supplier based on their ‘supplierID’. It returns a promise that resolves with an array of products. The function constructs a SQL query to select all products from the Products table where the ‘supplierID’ matches the provided parameter.

6.8.2.2.3 Controller

```

62  /**
63  * @desc Fetches a supplier and its products by supplier ID for user for selecting product
64  * @param {Object} req - Express request object
65  * @param {Object} req.params - Request parameters containing the supplier ID
66  * @param {number} req.params.supplierID - The ID of the supplier to fetch
67  * @param {Object} req.session - Session object containing user session data
68  * @param {Object} req.session.user - The current logged-in user's session data
69  * @param {Object} res - Express response object
70  * @returns {Promise<void>}
71  * @throws {Error} - Throws an error if there is an issue fetching the supplier or products
72  */
73  exports.showSelectProduct = async (req, res) => {
74
75  try {
76      const supplierId = req.params.supplierID;
77      const supplier = await userModel.getSupplierById(supplierId);
78      const products = await userModel.getProductsBySupplierId(supplierId);
79
80      // console.log(suppliers); // Debug output to confirm data fetching
81      console.log("Fetching supplier and its products with supplier ID:", supplierId);
82      res.render('User/selectproduct', {
83          user: req.session.user,
84          supplier: supplier,
85          products: products,
86      });
87  } catch (error) {
88      console.error('Failed to fetch supplier or products:', error);
89      res.status(500).render('error', { message: 'Failed to load supplier details and products.' });
90  }
91 };

```

Figure 6. 89 Controller for Displaying Select Product Page in ‘userController.js’

This controller function ensures that when a user requests to view the products of a specific supplier, the supplier’s details and its associated products are fetched from the database and displayed on the ‘selectproduct’ page.

6.8.2.2.4 View

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8" />
5      <meta
6          name="viewport"
7          content="width=device-width, initial-scale=1.0"
8      />
9      <title>Select Product</title>
10     <!-- Bootstrap CSS -->
11     <link
12         rel="stylesheet"
13         href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
14     />
15     <link
16         rel="stylesheet"
17         href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-icons/1.5.0/font/bootstrap-icons.min.css"
18     />
19     <!-- End of Bootstrap CSS -->
20     <link
21         rel="stylesheet"
22         href="/css/public.css"
23     />
24     <link
25         rel="stylesheet"
26         href="/css/selectproduct.css"
27     />
28     <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
29  </head>
30
31  <body>
32      <!-- Navigation Bar -->
33      <% include('../partials/navbar') %>
34      <!-- End of Navigation Bar -->
35
36      <!-- Left Side: Order Summary and Right Side: Select Product Section -->
37      <div class="container">
38          <div class="row vh-100 justify-content-center align-items-center">
39              <div class="row">
40                  <!-- Left Side: Order Summary -->
41                  <div class="col-md-4 my-1g-5">...
42              </div>
43              <!-- End of Left Side: Order Summary -->
44
45                  <!-- Right Side: Select Product -->
46                  <div class="col-md-8">...
47              </div>
48              <!-- End of Right Side: Select Product -->
49          </div>
50      </div>
51  </div>
52
53      <!-- End of Left Side: Order Summary and Right Side: Select Product Section -->
54
55      <!-- Partial Modals -->
56      <% include('../partials/usermodals') %> <%
57          include('../partials/signoutmodal') %> <%
58          include('../partials/uploadprofilephotomodal') %>
59      <!-- End of Partial Modals -->
60
```

Figure 6. 90 Overview of ‘selectproduct.ejs’

In the ‘selectproduct.ejs’ interface, users can select products. This interface is divided into two sections: the left side displays the order summary, and the right side lists the products available for selection. Client-side JavaScript is used to enhance user interaction with the webpage. For example, when a product is selected, it is immediately displayed on the left side in the order summary without requiring a page refresh.

```

40      <!-- Left Side: Order Summary -->
41      <div class="col-md-4 my-lg-5">
42          <h2>Order Summary</h2>
43          <div class="order-summary">
44              <ul class="list-group mb-3">
45                  <!-- Order Summary Items here -->
46                  <li
47                      class="list-group-item d-flex justify-content-between lh-sm"
48                  >
49                      <div>
50                          <h6 class="my-0">No Product here yet</h6>
51                          <small class="text-muted">RM xxx</small>
52                      </div>
53                      <span class="text-muted">Quantity</span>
54                  </li>
55              </ul>
56              <!-- Message Container -->
57              <div
58                  id="checkout-message"
59                  class="text-danger mt-3"
60              ></div>
61          </div>
62          <button
63              class="btn btn-primary w-100"
64              type="button"
65              onclick="proceedToCheckout()"
66          >
67              Proceed to Checkout
68          </button>
69      </div>
70      <!-- End of Left Side: Order Summary -->

```

Figure 6. 91 Order Summary Section in ‘selectproduct.ejs’

The button ‘Proceed to Checkout’ is placed in the order summary section. It is used for reviewing the selected item and quantity, thus proceeding to the place order page.

```

176      <!-- Client Side JS -->
177      <script>
178          // Restores the session state and sets the current supplier ID in sessionStorage when the document is loaded
179          document.addEventListener("DOMContentLoaded", () => { ...
180
181          // Handles the beforeunload event to prompt the user if there are unsaved order items
182          window.addEventListener("beforeunload", function (e) { ...
183
184          // Restores the order state from sessionStorage and updates the order summary
185          function restoreSessionState() { ...
186
187          // Updates the quantity of a product and saves it to sessionStorage
188          function updateQuantity(productId, supplierId, change) { ...
189
190          // Saves the order state to sessionStorage
191          function updateSessionStorage(productId, supplierId, quantity) { ...
192
193          // Updates the order summary display
194          function updateOrderSummary() { ...
195
196          // Removes a product from the order and updates the order summary
197          function deleteProduct(productId, supplierId) { ...
198
199          // Proceeds to the checkout page, encoding the order details in the URL
200          function proceedToCheckout() { ...
201
202          // Remove the navigation expected flag from sessionStorage on load
203          sessionStorage.removeItem("isNavigationExpected");
204      </script>
205      <!-- End of Client Side JS -->

```

Figure 6. 92 Overview of Client-Side JavaScript in ‘selectproduct.ejs’

These JavaScript functions in ‘selectproduct.ejs’ interact to manage order state, update the UI dynamically, and ensure a smooth product selection and checkout process. When the document loads,

the ‘DOMContentLoaded’ event restores the session state and sets the current supplier ID in ‘sessionStorage’. The ‘restoreSessionState’ function retrieves stored order details and updates product quantities accordingly.

The ‘updateQuantity’ function adjusts product quantities and calls ‘updateSessionStorage’ to save changes. ‘updateSessionStorage’ maintains order details in ‘sessionStorage’, ensuring persistence across page reloads.

The ‘updateOrderSummary’ function reads current order details and updates the order summary display. It dynamically generates a list of selected products, showing names, quantities, and total prices.

The ‘deleteProduct’ function allows users to remove products, updates session storage, and refreshes the order summary display.

```
326 // Proceeds to the checkout page, encoding the order details in the URL
327 function proceedToCheckout() {
328     //Proceed to checkout page
329     const order = JSON.parse(sessionStorage.getItem("order") || "{}");
330     const currentSupplierID = "<!-- supplier.supplierID -->";
331     const messageContainer = document.getElementById("checkout-message");
332
333     if (
334         !order[currentSupplierID] ||
335         Object.keys(order[currentSupplierID]).length === 0
336     ) {
337         // Display a message if no products are selected
338         messageContainer.textContent =
339             "No products selected. Please add products to your order before proceeding to checkout.";
340
341         // Set a timeout to clear the message after 2500 milliseconds (2.5 seconds)
342         setTimeout(function () {
343             messageContainer.textContent = ""; // Clear the text content of the message container
344         }, 2500);
345
346         return;
347     }
348
349     // Encode the order details and navigate to the place order page
350     const orderDetailsEncoded = encodeURIComponent(
351         JSON.stringify(order[currentSupplierID])
352     );
353     isNavigationExpected = true;
354     sessionStorage.setItem("isNavigationExpected", true);
355     window.location.href = `/user/placeorder?orderDetails=${orderDetailsEncoded}&supplierID=${currentSupplierID}`;
356 }
357
358 // Remove the navigation expected flag from sessionStorage on load
359 sessionStorage.removeItem("isNavigationExpected");
```

Figure 6. 93 Function ‘proceedToCheckout()’ in ‘selectproduct.ejs’

The ‘proceedToCheckout’ function checks if products are selected, displays a message if none are selected, and navigates to the Place Order page with encoded order details if items are present. This function saves the selected product information into a URL. If no products are selected, it prompts a message rejecting the request to proceed to checkout.

Once the request is successful, it directs the user to the URL /user/placeorder?orderDetails=\${orderDetailsEncoded}&supplierID=\${currentSupplierID}.

6.8.2.3 F005 Place Order

6.8.2.3.1 Route

```
46  /**
47  * @route GET /placeorder
48  * @desc Show place order page
49  * @access Private (User only)
50  */
51  router.get('/placeorder', isAuthenticated, userController.placeOrder);
```

Figure 6. 94 Route for Place Order Page Rendering in ‘userRoutes.js’

This route is used to display the place order page for authenticated users. When a user navigates to ‘/placeorder’, the ‘isAuthenticated’ middleware ensures they are logged in. If they are authenticated, the ‘placeOrder’ method from the ‘userController’ is called to render the appropriate view.

6.8.2.3.2 Model

```
110 /**
111  * @desc Fetches product details by product ID
112  * @param {number} productId - The ID of the product to retrieve details for
113  * @returns {Promise<Object>} - A promise that resolves with the product details
114  * @throws {Error} - Throws an error if there is an issue with the database query
115  */
116 fetchProductDetails: function(productId) {
117     return new Promise((resolve, reject) => {
118         // SQL query to retrieve product details by product ID
119         const query = `SELECT * FROM Products WHERE productID = ?`;
120         db.get(query, [productId], (err, row) => {
121             if (err) {
122                 console.error('Error fetching product details:', err);
123                 reject(err);
124             } else {
125                 resolve(row);
126             }
127         });
128     });
129 },
```

Figure 6. 95 Model for Fetching Product Details in ‘userModels.js’

The ‘fetchProductDetails’ function in ‘userModels.js’ is responsible for retrieving detailed information about a specific product from the database using its product ID. It is essential for ensuring that the order contains accurate and complete information about each product. It queries the database to retrieve detailed information for each product included in the order, which is then used to enrich the order details before rendering the confirmation page. This is required due to the utilization of temporary data structures in the URL.

```

132  /**
133  * @desc Creates a new order in the database
134  * @param {Object} orderData - The data of the order to be created
135  * @param {number} orderData.userID - The ID of the user placing the order
136  * @param {number} orderData.supplierID - The ID of the supplier fulfilling the order
137  * @param {string} orderData.date - The date the order is placed
138  * @param {string} orderData.deliveryDate - The date the order is to be delivered
139  * @param {number} orderData.totalPrice - The total price of the order
140  * @returns {Promise<number>} - A promise that resolves with the ID of the newly created order
141  * @throws {Error} - Throws an error if there is an issue with the database query
142 */
143 createOrder: function(orderData) {
144     return new Promise((resolve, reject) => {
145         // SQL query to insert a new order
146         const query = `
147             INSERT INTO Orders
148                 (userID, supplierID, date, deliveryDate, totalPrice, deliveryStatus)
149                 VALUES (?, ?, ?, ?, ?, ?)
150         `;
151         // Default the deliveryStatus to 'Pending'
152         db.run(query, [
153             orderData.userID,
154             orderData.supplierID,
155             orderData.date,
156             orderData.deliveryDate,
157             orderData.totalPrice,
158             'Pending' // Default status
159         ], function(err) {
160             if (err) {
161                 console.error("Failed to insert order due to error:", err);
162                 reject(err);
163             } else {
164                 console.log("Order successfully inserted with ID:", this.lastID);
165                 resolve(this.lastID);
166             }
167         });
168     });
169 },

```

Figure 6. 96 Model for Creating Order in ‘userModels.js’

```

72 /**
73 * @desc Creates a new order detail in the database
74 * @param {Object} detail - The data of the order detail to be created
75 * @param {number} detail.orderID - The ID of the order
76 * @param {number} detail.productID - The ID of the product
77 * @param {string} detail.productName - The name of the product
78 * @param {string} detail.productDescription - The description of the product
79 * @param {number} detail.quantity - The quantity of the product
80 * @param {number} detail.price - The price of the product
81 * @returns {Promise<number>} - A promise that resolves with the ID of the newly created order detail
82 * @throws {Error} - Throws an error if there is an issue with the database query
83 */
84 createOrderDetail: function(detail) {
85     return new Promise((resolve, reject) => {
86         // SQL query to insert a new order detail
87         const query = "INSERT INTO OrderDetails (orderID, productID, productName, productDescription, quantity, price) VALUES (?, ?, ?, ?, ?, ?)";
88         db.run(query, [detail.orderID, detail.productID, detail.productName, detail.productDescription, detail.quantity, detail.price], function(err) {
89             if (err) reject(err);
90             else resolve(this.lastID);
91         });
92     });
93 },

```

Figure 6. 97 Model for Creating Order Detail in ‘userModels.js’

Both functions are actually used to record the order and its details into the database. The ‘createOrder’ function inserts the data, ‘userID’, ‘supplierID’, ‘date’, ‘deliveryDate’, ‘totalPrice’, ‘deliveryStatus’ into ‘Orders’ table.

The function, ‘createOrderDetails’ record the ‘orderID’, ‘productID’, ‘productName’, ‘productDescription’, quantity, price into ‘OrderDetails’ table. It is used to record the information that is associated with the specific ‘orderID’ while considering the normalization practices.

6.8.2.3.3 Controller

```
94  /**
95   * @desc Processes the order placement by enriching the order details with product information and rendering the order confirmation page
96   * @param {Object} req - Express request object
97   * @param {Object} req.query - Query parameters containing order details and supplier ID
98   * @param {string} req.query.orderDetails - The encoded order details
99   * @param {number} req.query.supplierID - The ID of the supplier
100  * @param {Object} req.session - Session object containing user session data
101  * @param {Object} req.session.user - The current logged-in user's session data
102  * @param {Object} res - Express response object
103  * @returns {Promise<void>}
104  * @throws {Error} - Throws an error if there is an issue fetching the product or supplier details
105 */
106 exports.placeOrder = async (req, res) => {
107   const orderDetailsEncoded = req.query.orderDetails || "{}";
108   const orderDetails = JSON.parse(decodeURIComponent(orderDetailsEncoded));
109   const supplierID = req.query.supplierID; // Get supplier ID from query parameters
110   const currentDate = new Date().toISOString().slice(0, 10); // Get the current date in YYYY-MM-DD format
111
112   try {
113     const supplier = await userModel.getSupplierById(supplierID); // Fetch supplier details
114     const enrichedOrderDetails = await Promise.all(
115       Object.entries(orderDetails).map(async ([productId, quantity]) => {
116         const product = await userModel.fetchProductDetails(productId);
117         if (product) {
118           return {
119             ...product,
120             quantity: quantity,
121             totalPrice: product.productPrice * quantity
122           };
123         } else {
124           return { productId, name: "Product not found", price: 0, quantity };
125         }
126       })
127     );
128
129     res.render('user/placeorder', {
130       currentDate: currentDate,
131       orderDetails: enrichedOrderDetails,
132       user: req.session.user,
133       supplier: supplier // Pass the supplier details to the template
134     });
135   } catch (error) {
136     console.error('Failed to fetch product or supplier details:', error);
137     res.status(500).send("Error fetching product or supplier details.");
138   }
139 };


```

Figure 6. 98 Controller for Rendering Place Order Page in ‘userController.js’

This controller is mainly responsible for processing the order placement, enriching the order details with product information, and rendering the Place Order page. As the URL only stores the information like ‘supplierID’, ‘userID’, selected ‘productID’ and its ‘quantity’. The ‘placeOrder’ function then enriches the selected product information like ‘productName’ and so on information to display them on the interface.

By achieving this, it retrieves the encoded order details and supplier ID from the query parameters. It decodes and parses the order details, then fetches the supplier details using the supplier ID. For each product in the order, it retrieves detailed information from the database. This enriched order information is then used to render the ‘placeorder’ view template, displaying a comprehensive summary of the order

```

142 /**
143 * @desc Records an order by creating order and order detail entries in the database, and then interacts with the blockchain to place the order
144 * @param {Object} req - Express request object
145 * @param {Object} req.session - Session object containing user session data
146 * @param {Object} req.session.user - The current logged-in user's session data
147 * @param {number} req.session.user.id - The ID of the current user
148 * @param {Object} req.body - Request body containing the order details
149 * @param {number} req.body.supplierID - The ID of the supplier
150 * @param {string} req.body.orderDetails - JSON stringified order details
151 * @param {string} req.body.grandTotal - The total price of the order
152 * @param {string} req.body.deliveryDate - The delivery date of the order
153 * @param {Object} res - Express response object
154 * @returns {Promise<void>}
155 * @throws {Error} - Throws an error if there is an issue recording the order
156 */
157 exports.recordOrder = async (req, res) => {
158   if (!req.session.user) {
159     return res.status(403).send("No user session found, please login.");
160   }
161
162   const userID = req.session.user.id;
163   const { supplierID, orderDetails, grandTotal, deliveryDate } = req.body;
164   const currentDate = new Date().toISOString().slice(0, 10);
165
166   try {
167     const parsedOrderDetails = JSON.parse(orderDetails);
168     const orderID = await userModels.createOrder({
169       userID,
170       supplierID,
171       date: currentDate,
172       deliveryDate,
173       totalPrice: parseFloat(grandTotal)
174     });
175
176     await Promise.all(parsedOrderDetails.map(item =>
177       userModels.createOrderDetail({
178         orderID,
179         productID: item.productID,
180         productName: item.productName,
181         productDescription: item.productDescription,
182         quantity: item.quantity,
183         price: item.productPrice
184       })
185     ));
186
187     // Get user's Ethereum account
188     const accounts = await blockchain.web3.eth.getAccounts();
189
190     await blockchain.placeOrder({
191       userID: accounts[0],
192       supplierID,
193       deliveryDate,
194       totalPrice: parseFloat(grandTotal),
195       orderDetails: parsedOrderDetails.map(item => ({
196         orderID,
197         productID: item.productID,
198         productName: item.productName,
199         productDescription: item.productDescription,
200         quantity: item.quantity,
201         price: parseFloat(item.productPrice)
202       }))
203     });
204
205     res.redirect('/user/manageorder?orderPlaced=true');
206   } catch (error) {
207     console.error('Failed to record order:', error);
208     res.status(500).send("Failed to process order.");
209   }
210 };

```

Figure 6. 99 Controller for Recording Order in ‘userController.js’

The ‘recordOrder’ function in userController.js handles the recording of an order by creating order and order detail entries in the database and interacting with the blockchain to place the order. The main process starts by parsing the order details from the JSON string. Then, the order and order details is recorded by using ‘userModels.createOrder’ and ‘userModels.createOrderDetail’.

Besides that, it also call the ‘placeOrder’ method on the blockchain for passing the order details, total price, and other relevant information. This will be discussed further later.

Lastly, the controller will direct the user to the Manage Order page, once the order and its details are recorded to database.

6.8.2.3.4 View

```
30 <body>
31   <!-- Navigation Bar -->
32   <%-include('../partials/navbar') %>
33   <!-- End of Navigation Bar -->
34   <form
35     action="/user/placeorder"
36     method="POST"
37   >
38     <!-- Review and Submit Order Section -->
39     <div class="container">
40       <h2 class="py-5">Review and Submit Order</h2>
41
42       <h4 class="fw-bold">Order Details</h4>
43       <div>
44         <p>
45           <span class="label-bold">Supplier:</span>
46           <span class="text-muted"><%= supplier.companyName %></span>
47         </p>
48         <p>
49           <span class="label-bold">Date:</span>
50           <span class="text-muted"><%= currentDate %></span>
51         </p>
52       </div>
53
54     <!-- Select Delivery Date -->
55     <div class="row g-3 align-items-center"> ...
56   </div>
57   <div id="delivery-date-error" class="text-danger" style="display: none">
58     Please select the delivery date first
59   </div>
60   <!-- End of Select Delivery Date -->
61
62   <!-- Order Summary Table -->
63   <div class="container mt-3"> ...
64   </div>
65   <!-- End of Order Summary Table -->
66
67   <!-- Place Order Button -->
68   <div class="row justify-content-center"> ...
69   </div>
70   </div>
71 </form>
72 <!-- End of Place Order Button -->
73 <!-- End of Review and Submit Order Section -->
```

Figure 6. 100 Overview of 'placeorder.ejs'

The image provides a comprehensive code structure in 'placeorder.ejs'. It includes the sections such as, review and submit order section, date picker, confirmation modal and so on. The order summary is presented in tabular form to enhance the internal consistency of WoodChain in terms of front-end design.

The logic for users to select dates will enhance demand forecasting, making it easier for suppliers to prepare stock in advance.

```

223 | <!-- Client Side JS -->
224 | <script>
225 | // Initialize the date picker for the delivery date input
226 | $(document).ready(function () {
227 |   $("#delivery-date").datepicker({
228 |     format: "yyyy-mm-dd",
229 |     autoclose: true, // Automatically close the date picker after selecting a date
230 |     startDate: new Date(), // Set the minimum selectable date to today
231 |   });
232 |
233 | // Form submission event
234 | $("form").submit(function (event) {
235 |   event.preventDefault(); // Prevent the form from submitting immediately
236 |
237 |   var deliveryDate = $("#delivery-date").val();
238 |   if (!deliveryDate) {
239 |     $("#delivery-date-error").show(); // Show the error message
240 |
241 |     // Hide the error message after 2.5 seconds
242 |     setTimeout(function () {
243 |       $("#delivery-date-error").fadeOut("slow");
244 |     }, 2500);
245 |   } else {
246 |     // If the delivery date is selected, hide the error message
247 |     $("#delivery-date-error").hide();
248 |
249 |     // Show the confirmation modal
250 |     $("#confirmOrderModal").modal("show");
251 |   }
252 | });
253 |
254 | // Confirm order button event
255 | $("#confirmOrderButton").click(function () {
256 |   // Submit the form if confirmed
257 |   $("form")[0].submit();
258 | });
259 | });
260 |
261 | // Clear the sessionStorage when the form is submitted
262 | document.addEventListener("DOMContentLoaded", () => {
263 |   const placeorderForm = document.querySelector(
264 |     'form[action="/user/placeorder"]'
265 |   );
266 |   if (placeorderForm) {
267 |     placeorderForm.addEventListener("submit", () => {
268 |       sessionStorage.clear();
269 |     });
270 |   }
271 | });
272 | </script>
273 | <!-- End of Client Side JS -->

```

Figure 6. 101 Client-Side JavaScript in ‘placeorder.ejs’

This client-side JavaScript code in ‘placeorder.ejs’ manages the user interactions for placing an order. It initializes a date picker for the delivery date input field, setting the format to "yyyy-mm-dd", automatically closing the picker after a date is selected, and restricting the minimum selectable date to the current date.

The code handles the form submission event by preventing the default submission behavior and checking if a delivery date is selected. If the date is missing, an error message is displayed for 2.5 seconds. When the confirm order button in the modal is clicked, the form is submitted.

Additionally, the code listens for the DOMContentLoaded event to ensure that session storage is cleared when the form is submitted, removing any temporary data stored in session storage after the order is placed.

6.8.2.4 F014 Search Supplier

6.8.2.4.1 Route

```
28  /**
29  * @route POST /searchSuppliers
30  * @desc Search for suppliers by name
31  * @access Private (User only)
32  */
33  router.post('/searchSuppliers', isAuthenticated, userController.searchSuppliers);
```

Figure 6. 102 Route for Search Supplier Request Handling in ‘userRoutes.js’

This code snippet sets up a POST route in the ‘userRoutes.js’ file to handle requests for searching suppliers by their names. When a POST request is made to this endpoint, the ‘searchSuppliers’ method from ‘userController’ is invoked to perform the actual search operation. This route is essential for allowing users to search for suppliers in a secure manner, ensuring that only authorized users can perform the search action.

6.8.2.4.2 Model

```
32 /**
33 * @desc Searches for suppliers by company name
34 * @param {string} searchQuery - The search query used to find suppliers by name
35 * @returns {PromiseArray<Supplier>} - A promise that resolves with an array of suppliers matching the search query
36 * @throws {Error} - Throws an error if there is an issue with the database query
37 */
38 searchSuppliersByName: function(searchQuery) {
39   return new Promise((resolve, reject) => {
40     // SQL query to find suppliers by company name
41     const query = `
42       SELECT Users.userID, Users.companyName, Users.companyAddress, Users.profilePhoto, Suppliers.supplierID, Suppliers.supplierDescription
43       FROM Users
44       JOIN Suppliers ON Users.userID = Suppliers.userID
45       WHERE Users.companyName LIKE ?`;
46     db.all(query, [`%${searchQuery}%`], (err, rows) => {
47       if (err) {
48         console.error('Error searching suppliers:', err);
49         reject(err);
50       } else {
51         resolve(rows);
52       }
53     });
54   });
55 },
56 }
```

Figure 6. 103 Model for Searching Supplier in ‘userModels.js’

This function is designed to search for suppliers by their company name. It accepts a search query string and returns a promise that resolves to an array of suppliers whose company names match the search query. It retrieves data such as, ‘userID’, ‘companyName’, ‘companyAddress’, ‘profilePhoto’, ‘supplierID’, ‘supplierDescription’ to render the interface with sufficient information.

6.8.2.4.3 Controller

```
33  /**
34   * @desc Searches for suppliers by name and renders the user home page with the search results
35   * @param {Object} req - Express request object
36   * @param {Object} req.session - Session object containing user session data
37   * @param {Object} req.session.user - The current logged-in user's session data
38   * @param {Object} req.body - Request body containing the search query
39   * @param {string} req.body.searchQuery - The search query used to find suppliers by name
40   * @param {Object} res - Express response object
41   * @returns {Promise<void>}
42   * @throws {Error} - Throws an error if there is an issue searching for suppliers
43  */
44 exports.searchSuppliers = async (req, res) => {
45   try {
46     const { searchQuery } = req.body;
47     const suppliers = await userModels.searchSuppliersByName(searchQuery);
48
49     console.log('All searched suppliers retrieved successfully.');
50     res.render('user/userhome', {
51       user: req.session.user,
52       suppliers,
53       searchQuery // Pass the search query back to the template
54     });
55   } catch (error) {
56     console.error('Failed to search suppliers:', error);
57     res.status(500).render('error', { message: 'Failed to search suppliers.' });
58   }
59 }
```

Figure 6. 104 Controller for Searching Supplier in ‘userController.js’

The code snippet provided is a controller function in ‘userController.js’ that handles searching for suppliers by their name and rendering the user homepage with the search results. It applies the ‘userModels.searchSupplierByName’ to retrieve the information from the database. The ‘searchQuery’ is the data that is entered by the user through the search bar in the user homepage.

6.8.2.4.4 View

```
34  <!-- Search Supplier Bar -->
35  <div class="search-bar input-group mb-3 sticky-top">
36    <form
37      action="/user/searchSuppliers"
38      method="POST"
39      class="d-flex w-100"
40    >
41      <input
42        type="text"
43        name="searchQuery"
44        class="form-control"
45        placeholder="Search supplier"
46        aria-label="Search supplier"
47        aria-describedby="button-addon2"
48        value="<%= searchQuery || '' %>" 
49      />
50      <button
51        class="btn btn-outline-secondary"
52        type="submit"
53        id="button-addon2"
54      >
55        Search
56      </button>
57    </form>
58  </div>
59  <!-- End of Search Supplier Bar -->
```

Figure 6. 105 Search Supplier Bar in ‘userhome.ejs’

This is the section where the interface provides the search bar for user to key in the supplier company name for searching the preferred supplier.

```
61      <!-- Supplier List -->
62      <ul class="list-group">
63          <% if (suppliers && suppliers.length > 0) { %> <%
64          suppliers.forEach(supplier => { %>
65              <li class="list-group-item d-flex align-items-center">
66                  <div class="col-auto me-3">
67                      
72                  </div>
73                  <div class="col">
74                      <h5 class="mb-0"><%= supplier.companyName %></h5>
75                      <p class="mb-0 text-muted">
76                          <%= supplier.supplierDescription.length > 70 ?
77                              supplier.supplierDescription.substring(0, 70) + '...' :
78                              supplier.supplierDescription %>
79                      </p>
80                  </div>
81                  <a
82                      href="/user/selectproduct/<%= supplier.supplierID %>"
83                      class="stretched-link"
84                  ><%= supplier.supplierName %>
85              </a>
86          </li>
87          <% } ); %> <% } else { %>
88          <li class="list-group-item">No suppliers found.</li>
89          <% } %>
90      </ul>
91      <!-- End of Supplier List -->
92  
```

Figure 6. 106 Supplier List Section in ‘userhome.ejs’

The page can dynamically display the information ‘No suppliers found’ if there is no supplier found.

6.8.3 Supplier Functionality

6.8.3.1 Confirm Order (F008)

6.8.3.1.1 Route

```
108  /**
109   * @route POST /updateOrderStatus
110  * @desc Update the status of an order
111  * @access Private (Supplier only)
112 */
113 router.post('/updateOrderStatus', isAuthenticated, supplierController.updateOrderStatus);
```

Figure 6. 107 Route for Update Order Status Request Handling in ‘supplierRoutes.js’

The code snippet represents a route definition in ‘supplierRoutes.js’ for handling the update of an order’s status. The route is set to handle POST requests to the path ‘/updateOrderStatus’.

6.8.3.1.2 Model

```
266 /**
267  * @desc Updates the delivery status of an order to 'Confirmed' in the database
268  * @param {number} orderID - The ID of the order to be updated
269  * @returns {Promise<number>} - A promise that resolves with the number of rows affected by the update
270  * @throws {Error} - Throws an error if there is an issue with the database query
271 */
272 updateOrderStatus: function(orderID) {
273     return new Promise((resolve, reject) => {
274         const query = "UPDATE Orders SET deliveryStatus = 'Confirmed' WHERE orderID = ?";
275         db.run(query, [orderID], function(err) {
276             if (err) {
277                 console.error("Failed to update order status:", err);
278                 reject(err);
279             } else {
280                 console.log(`Updated ${this.changes} rows with orderID ${orderID}`);
281                 resolve(this.changes);
282             }
283         });
284     },
285 },
286 };
287 };
```

Figure 6. 108 Model for Updating Order Status in ‘supplierModels.js’

The function, ‘updateOrderStatus’, is used to update the ‘deliveryStatus’ of a specific order in the database. It sets the ‘deliveryStatus’ from a default value to ‘Confirmed’.

6.8.3.1.3 Controller

```
312 /**
313 * @desc Update order status in the database, and then interacts with the blockchain to record order status
314 * @param {Object} req - Express request object
315 * @param {Object} req.session - Session object containing user session data
316 * @param {Object} req.session.user - The current logged-in user's session data
317 * @param {Object} req.body - Request body containing order details
318 * @param {number} req.body.orderID - The ID of the order to be updated
319 * @param {string} req.body.status - The new status of the order
320 * @param {Object} res - Express response object
321 * @returns {Promise<void>}
322 * @throws {Error} - Throws an error if there is an issue updating the order status in the database or on the blockchain
323 */
324 exports.updateOrderStatus = async (req, res) => {
325   if (!req.session.user) {
326     return res.status(403).send("No user session found, please login.");
327   }
328
329   const { orderID, status } = req.body;
330
331   try {
332     // Get user's Ethereum account
333     const accounts = await blockchain.web3.eth.getAccounts();
334     const userAddress = accounts[0]; // Assumes the first account is the user's Ethereum address
335
336     if (!orderID || !status) {
337       console.error('Missing parameters for updating order status');
338       return res.status(400).json({ success: false, message: "Invalid request parameters" });
339     }
340
341     // Update order status in the database
342     const changes = await supplierModels.updateOrderStatus(orderID, status);
343
344
345     if (changes > 0) {
346       // Update order status on the blockchain
347       await blockchain.updateOrderStatus({ orderID, status, userID: userAddress });
348
349       res.json({ success: true });
350     } else {
351       res.status(404).json({ success: false, message: "Order not found or already confirmed" });
352     }
353   } catch (error) {
354     console.error("Error updating order status:", error);
355     res.status(500).json({ success: false, message: "Server error updating status", error: error.toString() });
356   }
357 }
```

Figure 6. 109 Controller for Updating Order Status in ‘supplierController.js’

The provided code is a controller function name ‘updateOrderStatus’ in ‘supplierController’. This function is responsible for updating the status of an order and recording the change on the blockchain.

It starts with examining the existence of user sessions. Then it will extract the ‘orderID’ and ‘state’ from the request body for the execution of the ‘supplierModels.updateOrderStatus’. The error handling mechanism has been applied to this function as well.

In a nutshell, it ensures the order status is updated both in the database and on the blockchain, maintaining data consistency and leveraging blockchain for additional security and transparency.

6.8.3.1.2 View

```

67      <!-- Orders Table -->
68  <div class="table-container">
69    <table
70      |   class="table table-hover table-borderless caption-top text-center align-center"
71    >
72      <caption>
73        |   Orders
74      </caption>
75      <thead>
76        <tr>
77          <th scope="col">Order ID</th>
78          <th scope="col">Company Name</th>
79          <th scope="col">Date</th>
80          <th scope="col">Delivery Date</th>
81          <th scope="col">Total</th>
82          <th scope="col">Status</th>
83        </tr>
84      </thead>
85      <tbody>
86        <% if (orders.length === 0) { %>
87          <tr>
88            <td
89              |   colspan="6"
90              |   class="text-center text-muted"
91            >
92              |   No orders found.
93            </td>
94          </tr>
95        <% } else { %> <% orders.forEach(function(order) { %>
96          <tr
97            |   onclick="handleRowClick('<%= order.orderID %>', '<%= order.deliveryStatus %>')"
98            |   style="cursor: pointer"
99          >
100            <td><%= order.orderID %></td>
101            <td><%= order.customerName %></td>
102            <td><%= order.date %></td>
103            <td><%= order.deliveryDate %></td>
104            <td>RM<%= order.totalPrice %></td>
105            <td>
106              <% if (order.deliveryStatus === 'Pending') { %>
107                <button
108                  |   class="btn btn-primary"
109                  |   onclick="showConfirmationModal('<%= order.orderID %>', event)"
110                >
111                  |   Confirm Order
112                </button>
113              <% } else { %>
114                <button
115                  |   class="btn btn-secondary"
116                  |   disabled
117                >
118                  |   Confirmed
119                </button>
120              <% } %>
121            </td>
122          </tr>
123          <% }); %> <% } %>
124        </tbody>
125      </table>
126    </div>
127    <!-- End of Orders Table -->
128  </div>
129  <!-- End of Header, Search Bar, and Orders Table Section -->

```

Figure 6. 110 Order Table Section in ‘supplierorder.ejs’

This code snippet represents the order table section in ‘supplierorder.ejs’. The design of interface is generally same as the ‘manageorder.ejs’ which use the tabular format to provide the order information that is associated with the specific user. The columns in the table include ‘Order ID’, ‘Company Name’, ‘Date’, ‘Delivery Date’, ‘Total’, and ‘Status’.

The button for updating order status will be displayed associated with every order. The button will only be clickable when it is in the ‘Pending’ status.

```

157  <!-- Confirm Order Confirmation Modal -->
158  <div
159   class="modal fade"
160   id="confirmationOrderModal"
161   tabindex="-1"
162   aria-labelledby="confirmationOrderModalLabel"
163   aria-hidden="true"
164  >
165  <div class="modal-dialog modal-lg modal-dialog-centered">
166   <div class="modal-content">
167    <div class="modal-header">
168     <h5
169      class="modal-title"
170      id="confirmationOrderModalLabel"
171     >
172      Confirm Order
173     </h5>
174     <button
175       type="button"
176       class="btn-close"
177       data-bs-dismiss="modal"
178       aria-label="Close"
179     ></button>
180   </div>
181   <div class="modal-body">
182    Are you sure you want to confirm the delivery of this order?
183   </div>
184   <div class="modal-footer">
185    <button
186      type="button"
187      class="btn btn-secondary"
188      data-bs-dismiss="modal"
189     >
190      Cancel
191     </button>
192    <button
193      type="button"
194      class="btn btn-primary"
195      id="confirmButton"
196     >
197      Confirm
198     </button>
199   </div>
200 </div>
201 </div>
202 </div>
203 <!-- End of Confirm Order Confirmation Modal -->

```

Figure 6. 111 Confirm Order Confirmation Modal in ‘supplierorder.ejs’

This modal will be triggered for the prevention of miss clicking the button.

```

279     // Confirm button click event listener
280     document
281       .getElementById("confirmButton")
282       .addEventListener("click", function () {
283         if (currentOrderID) {
284           updateStatus(currentOrderID);
285           const confirmationModal = bootstrap.Modal.getInstance(
286             document.getElementById("confirmationOrderModal")
287           );
288           confirmationModal.hide();
289         }
290       });
291
292     // Handles the row click event to show order details or open invoice
293     > window.handleRowClick = async function (orderID, orderStatus) { ...
294
295     // Updates the order status to "Confirmed"
296     function updateStatus(orderID) {
297       const status = "Confirmed";
298       fetch("/supplier/updateOrderStatus", {
299         method: "POST",
300         headers: {
301           "Content-Type": "application/json",
302         },
303         body: JSON.stringify({ orderID: orderID, status: status }), // Include status in the request body
304       })
305         .then((response) => {
306           if (!response.ok) {
307             throw new Error("Network response was not ok");
308           }
309           return response.json();
310         })
311         .then((data) => {
312           if (data.success) {
313             showMessage(
314               "Order status updated and recorded to blockchain successfully!",
315               "success"
316             );
317
318             // Update the button status immediately
319             currentButton.textContent = "Confirmed";
320             currentButton.classList.remove("btn-primary");
321             currentButton.classList.add("btn-secondary");
322             currentButton.disabled = true;
323
324             // Update the local status
325             const row = currentButton.closest("tr");
326             row
327               .querySelector("td:nth-child(6) button")
328               .classList.remove("btn-primary");
329             row
330               .querySelector("td:nth-child(6) button")
331               .classList.add("btn-secondary");
332             row.querySelector("td:nth-child(6) button").textContent =
333               "Confirmed";
334             row.querySelector("td:nth-child(6) button").disabled = true;
335           } else {
336             showMessage(
337               "Failed to update status: " + data.message,
338               "danger"
339             );
340           }
341         })
342         .catch((error) => {
343           console.error("Error updating status:", error);
344           showMessage("Error updating status: " + error, "danger");
345         });
346       }
347     }
348   }
349 
```

Figure 6. 112 Client-Side JavaScript in ‘supplierorder.ejs’

The script here manually manages the modal behaviour, improving smooth flow of interaction with the interface. It also handles the process of sending the request to correspond route with the ‘orderID’, ‘status’ in the request body.

When the order status is successfully updated, it will give feedback with the message, ‘Order status updated and recorded to blockchain successfully!’, to enhance the intuitive interaction in the system. This message will appear in the message container that is defined in the ‘supplierorder.ejs’.

6.8.3.2 F009 Edit Product

This functionality is composed of addition and deletion of product in the product list which associate with the certain supplier.

6.8.3.2.1 Route

```
15  /**
16   * @route GET /supplierhome
17   * @desc Show supplier home page with products
18   * @access Private (Supplier only)
19   */
20  router.get('/supplierhome', isAuthenticated, (req, res, next) => {
21    if (!req.session.user || req.session.user.type !== 'Supplier') {
22      return res.redirect('/login');
23    }
24    // If the user is authenticated and is a supplier, proceed to fetch supplier products
25    next(); // Pass control to the next middleware function, which is supplierController.showSupplierHome
26  }, supplierController.showSupplierHome);
```

Figure 6. 113 Route for Supplier Homepage Rendering in 'supplierRoutes.js'

This is a code snippet in 'userRoutes.js' file. It helps to show the user homepage for selecting supplier with the URL 'http://localhost:3000/supplier/supplierhome'. As there are a few accessibility roles in WoodChain, this route also provides a mechanism for ensuring that the session user exists, and their type is 'Supplier'. So, only the 'Supplier' can access the user homepage.

```
67  /**
68   * @route POST /updateProduct
69   * @desc Update an existing product
70   * @access Private (Supplier only)
71   */
72  router.post('/updateProduct', isAuthenticated, (req, res, next) => {
73    upload(req, res, (err) => {
74      if (err) {
75        console.error('Upload Error:', err);
76        res.status(500).render('supplier/supplierhome', { msg: err });
77      } else {
78        next();
79      }
80    });
81  }, supplierController.updateProduct);
```

Figure 6. 114 Route for Updating Product Information Request Handling in 'supplierRoutes.js'

This is the route used to handle the 'updateProduct' request to the '/updateProduct'. This route, accessible only to authenticated suppliers, ensures that users are logged in by using the 'isAuthenticated' middleware. It utilizes the upload middleware to handle file uploads. If the file upload is successful, the next function is called to proceed to the 'supplierController.updateProduct' function.

```
84  /**
85   * @route POST /deleteProduct
86   * @desc Delete a product
87   * @access Private (Supplier only)
88   */
89  router.post('/deleteProduct', isAuthenticated, supplierController.deleteProduct);
```

Figure 6. 115 Route for Deleting Product Request Handling in 'supplierRoutes.js'

The code snippet provided defines a POST route for deleting products in the supplierRoutes.js file.

6.8.3.2.2 Model

```

7   /**
8    * @desc Retrieves all active products associated with the specific supplier
9    * @param {number} supplierID - The ID of the supplier whose products are to be fetched
10   * @returns {Promise<Array>} - A promise that resolves with an array of products
11   * @throws {Error} - Throws an error if there is an issue with the database query
12   */
13  getSupplierProducts: function(supplierID) {
14    return new Promise((resolve, reject) => {
15      // SQL query to fetch all active products for a given supplier
16      const query = `
17        SELECT * FROM Products
18        WHERE supplierID = ? AND isActive = 1
19      `;
20      db.all(query, [supplierID], (err, products) => { // Make sure supplierId is correctly passed
21        if (err) {
22          reject(err);
23        } else {
24          resolve(products);
25        }
26      });
27    },
28  },

```

Figure 6. 116 Model for Retrieving Product in ‘supplierModels.js’

This function retrieves all the information from the ‘Products’ table where the value of attribute ‘isActive’ is equal to 1. The retrieved information will depend on the ‘supplierID’. The function will be used in the controller for displaying all the undeleted products on the supplier homepage.

```

106  /**
107   * @desc Adds a new product for a specific supplier
108   * @param {number} supplierID - The ID of the supplier adding the product
109   * @param {string} productName - The name of the product
110   * @param {string} productDescription - The description of the product
111   * @param {number} productPrice - The price of the product
112   * @param {string} productImage - The image of the product
113   * @returns {Promise<number>} - A promise that resolves with the ID of the newly added product
114   * @throws {Error} - Throws an error if there is an issue with the database query
115   */
116  addProduct: function(supplierID, productName, productDescription, productPrice, productImage) {
117    return new Promise((resolve, reject) => {
118      // SQL query to insert a new product
119      const query = `
120        INSERT INTO Products (supplierID, productName, productDescription, productPrice, productImage)
121        VALUES (?, ?, ?, ?, ?)
122      `;
123      db.run(query, [supplierID, productName, productDescription, productPrice, productImage], function(err) {
124        if (err) {
125          reject(err);
126        } else {
127          resolve(this.lastID); // returns the ID of the last inserted row
128        }
129      });
130    },
131  },

```

Figure 6. 117 Model for Adding Product in ‘supplierModels.js’

This model is used to insert data such as, ‘supplierID’, ‘productName’, ‘productDescription’, ‘productPrice’, and ‘productImage’ into ‘Products’ table. This model is crucial for adding a product to the database.

```

178  /**
179  * @desc Soft deletes a product by setting its isActive status to 2
180  * @param {number} productId - The ID of the product to delete
181  * @returns {Promise<number>} - A promise that resolves with the number of rows affected
182  * @throws {Error} - Throws an error if there is an issue with the database query
183  */
184  deleteProduct: function(productId) {
185      return new Promise((resolve, reject) => {
186          // SQL query to soft delete a product by setting its isActive status to 2
187          const query = `
188              UPDATE Products
189              SET isActive = 2
190              WHERE productID = ?
191          `;
192          db.run(query, [productId], function(err) {
193              if (err) {
194                  reject(err);
195              } else {
196                  resolve(this.changes);
197              }
198          });
199      });
200  },

```

Figure 6. 118 Model for Deleting Product in ‘supplierModels.js’

The ‘deleteProduct’ function will set the ‘isActive’ value to 2 based on the selected product. This is also known as the soft deletion of information. It does not directly delete or remove the product from the database to ensure that the other table’s information will not be affected.

6.8.3.2.3 Controller

```

51  /**
52  * @desc Renders the supplier home page with the supplier's products and description
53  * @param {Object} req - Express request object
54  * @param {Object} req.session - Session object containing user session data
55  * @param {Object} req.session.user - The current logged-in user's session data
56  * @param {number} req.session.user.supplierID - The ID of the supplier
57  * @param {Object} res - Express response object
58  * @returns {Promise<void>}
59  * @throws {Error} - Throws an error if there is an issue fetching products or supplier description
60  */
61 exports.showSupplierHome = async (req, res) => {
62     try {
63         const supplierID = req.session.user.supplierID;
64         const products = await supplierModels.getSupplierProducts(supplierID);
65         const description = await supplierModels.getSupplierDescription(supplierID);
66
67         console.log('All products retrieved successfully.');
68         res.render('supplier/supplierhome', {
69             user: req.session.user,
70             products,
71             description,
72             searchQuery: '' // Add default empty search query
73         });
74     } catch (error) {
75         console.error('Failed to fetch products:', error);
76         res.status(500).send('Failed to load products. Error: ' + error.message);
77     }
78 };

```

Figure 6. 119 Controller for Displaying Supplier Homepage in ‘supplierController.js’

The controller function plays the role of rendering the supplier homepage, which includes the supplier's products and description. It asynchronously fetches the supplier's products and description from the database by calling the respective methods from the 'supplierModels'.

```

8  /**
9   * @desc Configures storage settings for Multer
10  * - Sets destination folder for profile photos
11  * - Sets a unique filename using the current timestamp
12  */
13 const storage = multer.diskStorage({
14   destination: './public/uploads/productphoto/',
15   filename: function(req, file, cb) {
16     cb(null, 'product-' + Date.now() + path.extname(file.originalname));
17   }
18 });
19
20 /**
21  * @desc Configures Multer middleware for file upload
22  * - Uses the specified storage configuration
23  * - Filters files to allow only JPEG, JPG, and PNG formats
24  */
25 const upload = multer({
26   storage: storage,
27   limits: { fileSize: 1000000000 }, // Limit file size to 1MB
28   fileFilter: function(req, file, cb) {
29     checkFileType(file, cb);
30   }
31 }).single('productPhoto');
32
33 /**
34  * @desc Checks the file type to ensure it is a valid image format (JPEG, JPG, PNG)
35  * @param {Object} file - The file to be uploaded
36  * @param {Function} cb - Callback function to indicate if the file type is valid
37  */
38 function checkFileType(file, cb) {
39   const filetypes = /jpeg|jpg|png/;
40   const extname = filetypes.test(path.extname(file.originalname).toLowerCase());
41   const mimetype = filetypes.test(file.mimetype);
42
43   if (mimetype && extname) {
44     return cb(null, true);
45   } else {
46     cb('Error: Only JPEG, JPG, and PNG files are allowed!');
47   }
48 }

```

Figure 6. 120 Setting Up Mutler for Product Photo Upload in 'supplierController.js'

It first sets up the Mutler middleware for handling file uploads, especially for product photos. Then, the uploaded files will be stored to './public/uploads/productphoto/'. The filename will then be set with an unique filename.

Besides that, the function, in line25-31, is used to set up the Mutler Middleware. It sets the storage, limit storage of file, 'checkFileType', and specifies that only a single file named will be uploaded.

The 'checkFileType' is a mechanism to ensure that only the files with valid image formats such as, JPEG, JPG, and PNG are allowed to choose.

```

183 /**
184 * @desc Updates an existing product for the supplier and redirects to the supplier home page
185 * @param {Object} req - Express request object
186 * @param {Object} req.body - Request body containing the product details
187 * @param {number} req.body.product_id - The ID of the product to update
188 * @param {string} req.body.product_name - The new name of the product
189 * @param {string} req.body.product_description - The new description of the product
190 * @param {number} req.body.product_price - The new price of the product
191 * @param {Object} req.file - File object containing the uploaded file data (if any)
192 * @param {string} req.file.filename - The filename of the uploaded product image (if any)
193 * @param {Object} res - Express response object
194 * @returns {Promise<void>}
195 * @throws {Error} - Throws an error if there is an issue updating the product
196 */
197 exports.updateProduct = async (req, res) => {
198   const { product_id, product_name, product_description, product_price } = req.body;
199   let productImage; // Initialize without setting a default
200
201   // Check if a new file is uploaded
202   if (req.file) {
203     productImage = req.file.filename; // Only set if new image is uploaded
204   }
205
206   try {
207     // Update product without changing the image if no new image is provided
208     const result = await supplierModels.updateProduct(product_id, product_name, product_description, product_price, productImage);
209     if (result > 0) {
210       console.log('Update successful for product ID:', product_id);
211       res.redirect('/supplier/supplierhome');
212     } else {
213       console.log('No product found to update for product ID:', product_id);
214       res.status(404).send("Product not found");
215     }
216   } catch (error) {
217     console.error('Failed to update product:', error);
218     res.status(500).send('Failed to update product. Error: ' + error.message);
219   }
220 };

```

Figure 6. 121 Controller for Updating Product Information in ‘supplierController.js’

The code snippet is designed to update an existing product for a supplier and then redirect the user back to the supplier homepage. It works by first extracting the necessary product details from the request body and checking if a new product image has been uploaded. It means that the product image will remain the same if the supplier does not upload a new image when updating the product information.

```

223 /**
224 * @desc Soft deletes a product by setting its isActive status to 2 and redirects to the supplier home page
225 * @param {Object} req - Express request object
226 * @param {Object} req.body - Request body containing the product ID
227 * @param {number} req.body.product_id - The ID of the product to delete
228 * @param {Object} res - Express response object
229 * @returns {Promise<void>}
230 * @throws {Error} - Throws an error if there is an issue deleting the product
231 */
232 exports.deleteProduct = async (req, res) => {
233   const { product_id } = req.body;
234
235   try {
236     const result = await supplierModels.deleteProduct(product_id);
237     if (result > 0) {
238       console.log('Delete successful for product ID:', product_id);
239       res.redirect('/supplier/supplierhome');
240     } else {
241       console.log('No product found to delete for product ID:', product_id);
242       res.status(404).send("Product not found");
243     }
244   } catch (error) {
245     console.error('Failed to delete product:', error);
246     res.status(500).send('Failed to delete product. Error: ' + error.message);
247   }
248 };

```

Figure 6. 122 Controller for Deleting Product in ‘supplierController.js’

The controller function aims to delete the product with the ‘supplierModels.deleteProduct’.

6.8.3.2.4 View

```

25 <body style="overflow: hidden">
26   <!-- Navigation Bar -->
27   <% include('../partials/navbar') %>
28   <!-- End of Navigation Bar -->
29
30   <!-- Header, Search Bar, Add Product, and Product List Section -->
31   <div class="container">
32     <!-- Header and Search Bar -->
33     <h1 class="header sticky-top">Product List</h1>
34     <!-- Search Product Bar -->
35   <div class="search-bar input-group mb-3 sticky-top">...
36   </div>
37   <!-- End of Search Product Bar -->
38   <!-- End of Header and Search Bar -->
39
40   <!-- Add Product -->
41   <div class="row justify-content-end pb-3">
42     <div class="col-auto">
43       <a href="#" class="link-secondary text-decoration-underline text-primary" data-bs-toggle="modal" data-bs-target="#addproductModal">
44         >
45         Add Product
46       </a>
47     </div>
48   </div>
49   <!-- End of Add Product -->
50
51   <!-- Product List -->
52   <ul class="list-group">
53     <% if (products && products.length > 0) { %> <%
54     products.forEach(function(product) { %>
55
56       <li class="list-group-item d-flex align-items-center" data-bs-toggle="modal" data-bs-target="#manageProductModal<%= product.productID %>">
57         <div class="col-auto me-3">
58           
59         </div>
60         <div class="col">
61           <h5 class="mb-0"><%= product.productName %></h5>
62           <p class="mb-0 text-muted">
63             <%= product.productDescription.length > 70 ?
64               product.productDescription.substring(0, 70) + '...' :
65               product.productDescription %>
66           </p>
67         </div>
68         <a href="#" class="stretched-link" ></a>
69       <div class="row justify-content-end">
70         <div class="col-auto text-center"></div>
71         <span>RM <%= product.productPrice %> / each</span>
72       </div>
73     </li>
74     <% }); %> <% } else { %>
75       <li class="list-group-item">No products found.</li>
76     <% } %>
77   </ul>
78   <!-- End of Product List -->
79 </div>
80   <!-- End of Header, Search Bar, Add Product, and Product List Section -->
81
82   <!-- Modal -->
83   <% products.forEach(function(product) { %> <%
84     include('../partials/manageproductmodal', {product: product}) %><% }) ; %>
85   <%-include('../partials/suppliermodals') %>
86   <%-include('../partials/signoutmodal') %>
87   <%-include('../partials/uploadprofilephotomodal') %>
88   <!-- End of Modal -->
89
90   <!-- Bootstrap JS -->
91   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
92 </body>

```

Figure 6. 123 Overview of ‘supplierhome.ejs’

This code represents the interface design of supplier homepage. It includes a search bar, add product button and the product list. The partials that are involved in this page include ‘navbaar.ejs’,

‘manageproductmodal.ejs’, ‘suppliermodals.ejs’, ‘signoutmodal.ejs’, and ‘uploadprofilephotomodal.ejs’.

```

1  <!-- Manage Product Modal -->
2  <div
3    class="modal fade"
4    id="manageProductModal<%= product.productID %>"
5    tabindex="-1"
6    aria-labelledby="productModalLabel"
7    aria-hidden="true"
8  >
9    <div class="modal-dialog modal-dialog-centered modal-lg">
10      <div class="modal-content">
11        <div class="modal-header">
12          <h5
13            class="modal-title"
14            id="productModalLabel"
15          >
16            Manage Your Product
17          </h5>
18          <button
19            type="button"
20            class="btn-close"
21            data-bs-dismiss="modal"
22            aria-label="Close"
23          ></button>
24        </div>
25        <div class="modal-body">
26          <!-- Update Product Form -->
27          <form
28            method="POST"
29            action="/supplier/updateProduct"
30            enctype="multipart/form-data"
31          >
32            <input
33              type="hidden"
34              name="product_id"
35              value="<% product.productID %>">
36            />
37            <!-- Product Name -->
38            <div class="mb-3">...
39            </div>
40            <!-- Product Description -->
41            <div class="mb-3">...
42            </div>
43            <!-- Price -->
44            <div class="mb-3">...
45            </div>
46            <!-- Image -->
47            <div class="mb-3">...
48            </div>
49          </div>
50          <div class="modal-footer">
51            <div>
52              <button
53                type="submit"
54                class="btn btn-primary"
55              >
56                Update Product
57              </button>
58            </div>
59          </div>
60        </form>
61
62        <!-- Delete Product Form -->
63        <form
64          method="POST"
65          action="/supplier/deleteProduct"
66        >
67          <input
68            type="hidden"
69            name="product_id"
70            value="<% product.productID %>">
71          />
72          <button
73            type="submit"
74            class="btn btn-danger"
75          >
76            Delete Product
77          </button>
78        </form>
79      </div>
80    </div>
81  </div>
82  <!-- End of Manage Product Modal -->
```

Figure 6. 124 Manage Product Modal in ‘manageproductmodal.ejs’

This modal displays the product information and allows the supplier to interact with it. No matter if the supplier wants to update product information or delete a product. This modal will disappear for the

action selection. As the Bootstrap is used, it is same with the signup and login pages in which the fill is missing and so on, it will prompt the supplier to fill it with correct format.

```

142 <!-- Update Product Confirmation Modal -->
143 <div
144   class="modal fade"
145   id="confirmUpdateModal<%= product.productID %>"
146   tabindex="-1"
147   aria-labelledby="confirmUpdateModalLabel"
148   aria-hidden="true"
149 >
150   <div class="modal-dialog modal-dialog-centered modal-lg">
151     <div class="modal-content">
152       <div class="modal-header">
153         <h5
154           class="modal-title"
155           id="confirmUpdateModalLabel"
156         >
157           Confirm Update
158         </h5>
159         <button
160           type="button"
161           class="btn-close"
162           data-bs-dismiss="modal"
163           aria-label="Close"
164         ></button>
165       </div>
166       <div class="modal-body">
167         Are you sure you want to update this product?
168       </div>
169       <div class="modal-footer">
170         <button
171           type="button"
172           class="btn btn-secondary"
173           data-bs-dismiss="modal"
174         >
175           Cancel
176         </button>
177         <button
178           type="button"
179           class="btn btn-primary"
180           id="confirmUpdateButton<%= product.productID %>"
181         >
182           Confirm
183         </button>
184       </div>
185     </div>
186   </div>
187 </div>
188 <!-- End of Update Product Confirmation Modal -->
189
190 <!-- Delete Product Confirmation Modal -->
191 <div
192   class="modal fade"
193   id="confirmDeleteModal<%= product.productID %>"
194   tabindex="-1"
195   aria-labelledby="confirmDeleteModalLabel"
196   aria-hidden="true"
197 >
198   <div class="modal-dialog modal-dialog-centered">
199     <div class="modal-content">
200       <div class="modal-header">
201         <h5 class="modal-title" id="confirmDeleteModalLabel">Confirm Delete</h5>
202         <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
203       </div>
204       <div class="modal-body">
205         Are you sure you want to delete this product?
206       </div>
207       <div class="modal-footer">
208         <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Cancel</button>
209         <button type="button" class="btn btn-danger" id="confirmDeleteButton<%= product.productID %>">Delete</button>
210       </div>
211     </div>
212   </div>
213 </div>
214 <!-- End of Delete Product Confirmation Modal -->
```

Figure 6. 125 Confirmation Modals in ‘manageproductmodal.ejs’

The image contains both confirmation modal for updating and deleting the product. It will appear based on which button is selected in manage product modal. Once the action is confirmed, the request will be posted to the correct route immediately through the client-side JavaScript.

```
216 <!-- Client Side JS -->
217 <script>
218 // Update the character count for the product description field as the user types
219 // Function for calculate the product description word count
220 > document.getElementById("product_description<%= product.productID %>").addEventListener('input', function () { ...
225
226 // Run the following when the DOM is fully loaded
227 // Function for managing process of updating products within a modal interface on a webpage
228 > document.addEventListener("DOMContentLoaded", function () { ...
301
302 // Run the following when the DOM is fully loaded
303 // Function for deleting product
304 > document.addEventListener("DOMContentLoaded", function () { ...
348 </script>
349 <!-- End of Client Side JS -->
```

Figure 6. 126 Client-Side JavaScript in 'manageproductmodal.ejs'

These scripts here are used to manage the modal behaviour and edit product process. For the update product form, it tracks the character count for product description, validates the product description to ensure it's not empty or only whitespace, displays a confirmation modal before submitting the form.

For the delete product form, it displays a confirmation modal before submitting the form to delete a product. It ensures the editing product process is user-friendly for the supplier. For instance, the manage product modal will automatically display editable product information. In case of accidentally closing the modal, it will remain the information until the page is refreshed or the form is submitted.

6.8.3.3 F010 Add Product

6.8.3.3.1 Route

```
51  /**
52  * @route POST /addProduct
53  * @desc Add a new product
54  * @access Private (Supplier only)
55  */
56 router.post('/addProduct', isAuthenticated, (req, res, next) => {
57   upload(req, res, (err) => {
58     if (err) {
59       res.status(500).render('supplier/supplierhome', { msg: err });
60     } else {
61       next();
62     }
63   });
64 }, supplierController.addProduct);
```

Figure 6. 127 Route for Adding Product Request Handling in ‘supplierRoutes.js’

This is the route used to handle the ‘addProduct’ request to the ‘/addProduct’. This route, accessible only to authenticated suppliers, ensures that users are logged in by using the ‘isAuthenticated’ middleware. It utilizes the upload middleware to handle file uploads. If the file upload is successful, the next function is called to proceed to the ‘supplierController.addProduct’ function.

6.8.3.3.2 Model

```
106 /**
107 * @desc Adds a new product for a specific supplier
108 * @param {number} supplierID - The ID of the supplier adding the product
109 * @param {string} productName - The name of the product
110 * @param {string} productDescription - The description of the product
111 * @param {number} productPrice - The price of the product
112 * @param {string} productImage - The image of the product
113 * @returns {Promise<number>} - A promise that resolves with the ID of the newly added product
114 * @throws {Error} - Throws an error if there is an issue with the database query
115 */
116 addProduct: function(supplierID, productName, productDescription, productPrice, productImage) {
117   return new Promise((resolve, reject) => {
118     // SQL query to insert a new product
119     const query = `
120       INSERT INTO Products (supplierID, productName, productDescription, productPrice, productImage)
121       VALUES (?, ?, ?, ?, ?)
122     `;
123     db.run(query, [supplierID, productName, productDescription, productPrice, productImage], function(err) {
124       if (err) {
125         reject(err);
126       } else {
127         resolve(this.lastID); // returns the ID of the last inserted row
128       }
129     });
130   });
131 },
```

Figure 6. 128 Model for Adding Product in ‘supplierModels.js’

The model function is used to allow the user to insert the new product into ‘Products’ table. The data that will be manipulated by this function includes ‘supplierID’, ‘productName’, ‘productDescription’, ‘productPrice’, and ‘productImage’.

6.8.3.3.3 Controller

```
145 /**
146 * @desc Adds a new product for the supplier and redirects to the supplier home page
147 * @param {Object} req - Express request object
148 * @param {Object} req.session - Session object containing user session data
149 * @param {Object} req.session.user - The current logged-in user's session data
150 * @param {number} req.session.user.supplierID - The ID of the supplier
151 * @param {Object} req.body - Request body containing the product details
152 * @param {string} req.body.product_name - The name of the product
153 * @param {string} req.body.product_description - The description of the product
154 * @param {number} req.body.product_price - The price of the product
155 * @param {Object} req.file - File object containing the uploaded file data
156 * @param {string} req.file.filename - The filename of the uploaded product image
157 * @param {Object} res - Express response object
158 * @returns {Promise<void>}
159 * @throws {Error} - Throws an error if there is an issue adding the product
160
161    orts.addProduct = async (req, res) => {
162         const supplierID = req.session.user.supplierID;
163         const { product_name, product_description, product_price } = req.body;
164
165         let productImage = ''; // Default image if none uploaded
166
167         // Check if req.file is defined and if a file was uploaded
168         if (req.file && req.file.filename) {
169             productImage = req.file.filename; // Set productImage to uploaded filename
170         }
171
172         try {
173             const productID = await supplierModels.addProduct(supplierID, product_name, product_description, product_price, productImage);
174             console.log('Product added with ID:', productID);
175             res.redirect('/supplier/supplierhome');
176         } catch (error) {
177             console.error('Failed to add product:', error);
178             res.status(500).send('Failed to add product. Error: ' + error.message);
179     }
```

Figure 6. 129 Controller for Adding Product in 'supplierController.js'

This code snippet is an asynchronous controller function designed to handle adding a new product for a supplier. It processes the product details, handles file uploads, and interacts with the model to insert the new product into the database.

As the controllers which have been implemented and discussed before, it starts with extracting the necessary product details from the request, handling optional file uploads for the product image, and interacts with the 'supplierModels.addProduct'

6.8.3.3.1 View

```
62     <!-- Add Product -->
63     <div class="row justify-content-end pb-3">
64         <div class="col-auto">
65             <a
66                 href="#"
67                 class="link-secondary text-decoration-underline text-primary"
68                 data-bs-toggle="modal"
69                 data-bs-target="#addproductModal"
70             >
71                 Add Product
72             </a>
73         </div>
74     </div>
75     <!-- End of Add Product -->
```

Figure 6. 130 Add Product Section in 'supplierhome.ejs'

This code snippet provides the interface for supplier to trigger the add product process. Once the 'Add Product' is clicked, the add product modal will then be displayed.

```

145  <!-- Add Product Modal -->
146  > <div...
147  </div>
148  <!-- End of Add Product Modal -->
149
150  <!-- Add Product Confirmation Modal -->
151  <div
152  |   class="modal fade"
153  |   id="confirmationModal"
154  |   tabindex="-1"
155  |   aria-labelledby="confirmationModalLabel"
156  |   aria-hidden="true"
157  >
158  |   <div class="modal-dialog modal-dialog-centered">
159  |   |   <div class="modal-content">
160  |   |   |   <div class="modal-header">
161  |   |   |   |   <h5
162  |   |   |   |   |   class="modal-title"
163  |   |   |   |   |   id="confirmationModalLabel"
164  |   |   |   |   >
165  |   |   |   |   |   Confirm Add Product
166  |   |   |   |   </h5>
167  |   |   |   |   <button
168  |   |   |   |   |   type="button"
169  |   |   |   |   |   class="btn-close"
170  |   |   |   |   |   data-bs-dismiss="modal"
171  |   |   |   |   |   aria-label="Close"
172  |   |   |   |   ></button>
173  |   |   |   </div>
174  |   |   <div class="modal-body">Are you sure you want to add this product?</div>
175  |   |   <div class="modal-footer">
176  |   |   |   <button
177  |   |   |   |   type="button"
178  |   |   |   |   class="btn btn-secondary"
179  |   |   |   |   data-bs-dismiss="modal"
180  |   |   |   >
181  |   |   |   |   Cancel
182  |   |   |   </button>
183  |   |   |   <button
184  |   |   |   |   type="button"
185  |   |   |   |   class="btn btn-primary"
186  |   |   |   |   id="confirmAddProductButton"
187  |   |   |   >
188  |   |   |   |   Yes, Add Product
189  |   |   |   </button>
190  |   |   </div>
191  |   </div>
192  </div>
193  <!-- End of Add Product Confirmation Modal -->

```

Figure 6. 131 Add Product and Confirmation Modal in ‘suppliermodals.ejs’

The interface of the add product modal is generally similar to the manage product modal. For example, it includes features such as description word count tracker and mechanisms to prevent the submission of erroneous data. Additionally, a confirmation modal is implemented as a mechanism to ensure that the user confirms the action before proceeding with any critical changes.

```

333 // Wait until the DOM content is fully loaded
334 document.addEventListener("DOMContentLoaded", function () {
335     // Function to update word count for a textarea
336     function updateWordCount(textareaId, counterId, maxLength) { ...
337
338     // Function to validate and submit the company description
339     function validateCompanyDescription() { ...
340
341
342     // Function to validate and show the confirmation modal before submitting the product form
343     function validateAndShowConfirmation() {
344         const form = document.getElementById("addProductForm");
345         const productDescription = document.getElementById("product_description");
346         const productDescriptionError = document.getElementById(
347             "product_description_error"
348         );
349
350         // Remove previous error message and class if any
351         productDescription.classList.remove("is-invalid");
352         if (productDescriptionError) {
353             productDescriptionError.remove();
354         }
355
356         // Check if the product description is empty or only whitespace
357         if (!productDescription.value.trim()) {
358             productDescription.classList.add("is-invalid");
359             const errorMessage = document.createElement("div");
360             errorMessage.id = "product_description_error";
361             errorMessage.className = "invalid-feedback";
362             errorMessage.innerText =
363                 "Product description cannot be empty or only whitespace.";
364             productDescription.parentNode.appendChild(errorMessage);
365             productDescription.focus();
366             return false;
367         }
368
369         // If the form is valid, show the confirmation modal
370         if (form.checkValidity()) {
371             const addProductModal = bootstrap.Modal.getInstance(
372                 document.getElementById("addproductModal")
373             );
374             const confirmationModal = new bootstrap.Modal(
375                 document.getElementById("confirmationModal")
376             );
377
378             // Hide the add product modal and show the confirmation modal
379             addProductModal.hide();
380             addProductModal._element.addEventListener(
381                 "hidden.bs.modal",
382                 function () {
383                     confirmationModal.show();
384                 },
385                 { once: true }
386             );
387
388             // When confirmation is clicked, submit the form
389             document.getElementById("confirmAddProductButton").onclick =
390                 function () {
391                     form.submit();
392                 };
393
394             // If the confirmation modal is closed without submitting, show the add product modal again
395             document
396                 .getElementById("confirmationModal")
397                 .addEventListener("hidden.bs.modal", function () {
398                     const confirmationModalInstance = bootstrap.Modal.getInstance(
399                         document.getElementById("confirmationModal")
400                     );
401                     if (
402                         confirmationModalInstance &&
403                         !confirmationModalInstance._isShown
404                     ) {
405                         addProductModal.show();
406                     }
407                 });
408             } else {
409                 form.reportValidity(); // Report form validity if invalid
410             }
411         }
412     }
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
}

```

Figure 6. 132 Client-Side JavaScript in ‘suppliermodals.ejs’

Based on the figure, the function ‘updateWordCount’ will be applied to the calculation word count of product and supplier description. Then, ‘validateAndShowConfirmation’ function is used to manage the modal behaviour and add product process. For example, it checks whether the form is valid before submitting the form.

6.8.3.4 F012 Add Company Description

6.8.3.4.1 Route

```
29  /**
30   * @route POST /updateDescription
31   * @desc Update supplier description
32   * @access Private (Supplier only)
33   */
34  router.post('/updateDescription', isAuthenticated, (req, res) => {
35    if (!req.session.user || req.session.user.type !== 'Supplier') {
36      return res.redirect('/login');
37    }
38    // Call the controller function to update the description
39    supplierController.updateSupplierDescription(req, res);
40  });
41
```

Figure 6. 133 Route for Update Supplier Description Request Handling in ‘supplierRoutes.js’

This route is used to handle the request of updating supplier company description. The request will finally send to ‘/updateDescription’ by using POST method.

6.8.3.4.2 Model

```
57  /**
58   * @desc Retrieves the description of a supplier from the database
59   * @param {number} supplierID - The ID of the supplier whose description is being retrieved
60   * @returns {Promise<string|null>} - A promise that resolves with the supplier's description or null if not found
61   * @throws {Error} - Throws an error if there is an issue with the database query
62   */
63  getSupplierDescription: function(supplierID) {
64    return new Promise((resolve, reject) => {
65      // SQL query to fetch the supplier's description
66      const query = `
67        SELECT supplierDescription FROM Suppliers
68        WHERE supplierID = ?`;
69
70      db.get(query, [supplierID], (err, row) => {
71        if (err) {
72          reject(err);
73        } else {
74          resolve(row ? row.supplierDescription : null);
75        }
76      });
77    });
78  },
```

Figure 6. 134 Model for Retrieving Supplier Description in ‘supplierModels.js’

This function is used to retrieve supplier company description based on the ‘supplierID’.

```
31  /**
32   * @desc Updates the description of a supplier in the database
33   * @param {number} supplierID - The ID of the supplier whose description is being updated
34   * @param {string} description - The new description of the supplier
35   * @returns {Promise<number>} - A promise that resolves with the number of rows affected
36   * @throws {Error} - Throws an error if there is an issue with the database query
37   */
38  updateSupplierDescription: function(supplierID, description) {
39    return new Promise((resolve, reject) => {
40      // SQL query to update the supplier's description
41      const query = `
42        UPDATE Suppliers
43        SET supplierDescription = ?
44        WHERE supplierID = ?`;
45
46      db.run(query, [description, supplierID], function(err) {
47        if (err) {
48          reject(err);
49        } else {
50          resolve(this.changes); // this.changes returns the number of rows affected
51        }
52      });
53    });
54  },
```

Figure 6. 135 Model for Updating Supplier Description in ‘supplierModels.js’

This code defines a function ‘updateSupplierDescription’ that updates the description of a supplier in the database. It uses an SQL query to set the new description for a supplier with a specific ‘supplierID’.

6.8.3.4.3 Controller

```
81  /**
82   * @desc Updates the description of a supplier
83   * @param {Object} req - Express request object
84   * @param {Object} req.session - Session object containing user session data
85   * @param {Object} req.session.user - The current logged-in user's session data
86   * @param {number} req.session.user.supplierID - The ID of the supplier
87   * @param {Object} req.body - Request body containing the new description
88   * @param {string} req.body.company_description - The new description of the supplier's company
89   * @param {Object} res - Express response object
90   * @returns {Promise<void>}
91   * @throws {Error} - Throws an error if there is an issue updating the supplier's description
92   */
93 exports.updateSupplierDescription = async (req, res) => {
94   const supplierID = req.session.user.supplierID;
95   const { company_description } = req.body;
96
97   try {
98     const result = await supplierModels.updateSupplierDescription(supplierID, company_description);
99     if (result > 0) {
100       console.log("Description updated successfully!");
101       res.redirect('/supplier/supplierhome');
102     } else {
103       res.status(404).send("Supplier not found");
104     }
105   } catch (error) {
106     console.error('Failed to update description:', error);
107     res.status(500).send('Failed to update description. Error: ' + error.message);
108   }
109};
```

Figure 6. 136 Controller for Updating Supplier Description in ‘supplierController.js’

The ‘updateSupplierDescription’ function in ‘supplierController.js’ is designed to update a supplier's description in the database. It retrieves the ‘supplierID’ from the session data and the new ‘company_description’ from the request body. The function then calls a model method to perform the update.

6.8.3.4.4 View

```
1  <!-- Supplier Profile Options Modal-->
2  > <div>...
66  </div>
67  <!-- End of Supplier Profile Options Modal -->
68
69  <!-- Add Company Description Modal -->
70  <div
71    class="modal fade"
72    id="addCompanyDescriptionModal"
73    tabindex="-1"
74    aria-labelledby="addCompanyDescriptionModalLabel"
75    aria-hidden="true"
76  >
77    <div class="modal-dialog modal-dialog-centered">
78      <div class="modal-content">
79        <div class="modal-header">
80          <h5
81            class="modal-title"
82            id="addCompanyDescriptionModalLabel"
83          >
84            Add Company Description
85          </h5>
86          <button
87            type="button"
88            class="btn-close"
89            data-bs-dismiss="modal"
90            aria-label="Close"
91          ></button>
92        </div>
93        <div class="modal-body">
94          <form
95            id="addCompanyDescriptionForm"
96            action="/supplier/updateDescription"
97            method="POST"
98          >
99            <div class="col-12">
100              <div class="form-floating mb-2">
101                <textarea
102                  class="form-control"
103                  name="company_description"
104                  id="company_description"
105                  placeholder="Company Description"
106                  required
107                  maxlength="300"
108                  oninput="updateWordCount('company_description', 'company_description_word_count', 300)"
109                >
110                <%= description %></textarea>
111                >
112                <label for="company_description">Company Description</label>
113                <div
114                  id="company_description_error"
115                  class="invalid-feedback"
116                ></div>
117              </div>
118              <div class="text-end">
119                <small id="company_description_word_count">0/300</small>
120              </div>
121            </div>
122            <div class="modal-footer">
123              <button
124                type="button"
125                class="btn btn-secondary"
126                data-bs-dismiss="modal"
127              >
128                Close
129              </button>
130              <button
131                type="button"
132                class="btn btn-primary"
133                onclick="validateCompanyDescription()"
134              >
135                Save Description
136              </button>
137            </div>
138          </form>
139        </div>
140      </div>
141    </div>
142  </div>
143  <!-- End of Add Company Description Modal -->
```

Figure 6. 137 Add Company Description Modal in ‘suppliermodals.ejs’

This code provides the interface of updating or adding company description. Since, there is no description will be displayed in ‘profileInformation.ejs’. This is the only part where the supplier can check, add, and update their company description.

```
333 // Wait until the DOM content is fully loaded
334 document.addEventListener("DOMContentLoaded", function () {
335     // Function to update word count for a textarea
336     function updateWordCount(textareaId, counterId, maxLength) {
337         const textarea = document.getElementById(textareaId);
338         const counter = document.getElementById(counterId);
339         const currentLength = textarea.value.length;
340         counter.textContent = `${currentLength}/${maxLength}`;
341     }
342
343     // Function to validate and submit the company description
344     function validateCompanyDescription() {
345         const companyDescription = document.getElementById("company_description");
346         const companyDescriptionError = document.getElementById(
347             "company_description_error"
348         );
349
350         // Remove previous error message and class if any
351         companyDescription.classList.remove("is-invalid");
352         companyDescriptionError.innerText = "";
353
354         // Check if the description is empty or only whitespace
355         if (!companyDescription.value.trim()) {
356             companyDescription.classList.add("is-invalid");
357             companyDescriptionError.innerText =
358                 "Company description cannot be empty or only whitespace.";
359             companyDescription.focus();
360             return false;
361         }
362
363         // Check if the description exceeds the maximum length
364         if (companyDescription.value.length > 300) {
365             companyDescription.classList.add("is-invalid");
366             companyDescriptionError.innerText =
367                 "Company description cannot exceed 300 characters.";
368             companyDescription.focus();
369             return false;
370         }
371
372         // If valid, submit the form
373         const form = document.getElementById("addCompanyDescriptionForm");
374         form.submit();
375     }
}
```

Figure 6. 138 Client-Side JavaScript in ‘suppliermodals.ejs’

Similar to the client-side JavaScript for adding products, this also manages the modal behaviour and form submission process.

6.8.3.5 F016 Search Product

6.8.3.5.1 Route

```
43  /**
44  * @route POST /searchProducts
45  * @desc Search products by name
46  * @access Private (Supplier only)
47  */
48 router.post('/searchProducts', isAuthenticated, supplierController.searchProducts);
```

Figure 6. 139 Route for Search Supplier Request Handling in ‘supplierRoutes.js’

This code snippet sets up a POST route in the ‘supplierRoutes.js’ file to handle requests for searching products by the product name. When a POST request is made to this endpoint, the ‘searchSuppliers’ method from ‘supplierController’ is invoked to perform the actual search operation.

6.8.3.5.2 Model

```
81 /**
82  * @desc Searches for products by name for a specific supplier
83  * @param {number} supplierID - The ID of the supplier whose products are being searched
84  * @param {string} productName - The name of the product to search for
85  * @returns {Promise<Array>} - A promise that resolves with an array of products matching the search query
86  * @throws {Error} - Throws an error if there is an issue with the database query
87 */
88 getProductsByName: function(supplierID, productName) {
89     return new Promise((resolve, reject) => {
90         // SQL query to find products by name for a specific supplier
91         const query = `
92             SELECT * FROM Products
93             WHERE supplierID = ? AND productName LIKE ? AND isActive = 1
94         `;
95         db.all(query, [supplierID, `%"${productName}"%`], (err, products) => {
96             if (err) {
97                 reject(err);
98             } else {
99                 resolve(products);
100            }
101        });
102    });
103}
```

Figure 6. 140 Model for Retrieving Product Name in ‘supplierModels.js’

This code defines a function ‘getProductsByName’ that searches for products by name for a specific supplier. It constructs an SQL query to find products that match the supplier ID and have a name similar to the provided product name. The query also ensures that the product is active. The function returns a Promise that resolves with an array of matching products or rejects with an error if the query fails.

6.8.3.5.3 Controller

```

112  /**
113   * @desc Searches for products by name for a specific supplier and renders the supplier home page with the search results
114   * @param {Object} req - Express request object
115   * @param {Object} req.session - Session object containing user session data
116   * @param {Object} req.session.user - The current logged-in user's session data
117   * @param {number} req.session.user.supplierID - The ID of the supplier
118   * @param {Object} req.body - Request body containing the search query
119   * @param {string} req.body.searchQuery - The search query used to find products by name
120   * @param {Object} res - Express response object
121   * @returns {Promise<void>}
122   * @throws {Error} - Throws an error if there is an issue searching for products
123 */
124 exports.searchProducts = async (req, res) => {
125   try {
126     const supplierID = req.session.user.supplierID;
127     const { searchQuery } = req.body;
128     const products = await supplierModels.getProductsByName(supplierID, searchQuery);
129     const description = await supplierModels.getSupplierDescription(supplierID);
130
131     console.log('All searched products retrieved successfully.');
132     res.render('supplier/supplierhome', {
133       user: req.session.user,
134       products,
135       description: description || '', // Add default empty description
136       searchQuery // Pass the search query back to the template
137     });
138   } catch (error) {
139     console.error('Failed to search products:', error);
140     res.status(500).send('Failed to search products. Error: ' + error.message);
141   }
142 };

```

Figure 6. 141 Controller for Searching Product Name in ‘supplierController.js’

The ‘searchProducts’ function in supplierController.js is designed to search for products by name for a specific supplier and render the supplier home page with the search results. It retrieves the ‘supplierID’ from the session data and the ‘searchQuery’ from the request body. The function calls model methods to perform the search and retrieve the supplier description. If successful, it renders the home page with the search results.

6.8.3.5.4 View

```

34      <!-- Search Product Bar -->
35      <div class="search-bar input-group mb-3 sticky-top">
36        <form
37          action="/supplier/searchProducts"
38          method="POST"
39          class="d-flex w-100"
40        >
41          <input
42            type="text"
43            name="searchQuery"
44            class="form-control"
45            placeholder="Search product"
46            aria-label="Search product"
47            aria-describedby="button-addon2"
48            value="<%= searchQuery %>" 
49          />
50          <button
51            class="btn btn-outline-secondary"
52            type="submit"
53            id="button-addon2"
54          >
55            Search
56          </button>
57        </form>
58      </div>
59      <!-- End of Search Product Bar -->
60      <!-- End of Header and Search Bar -->

```

Figure 6. 142 Search Product Bar in ‘supplierhome.ejs’

The supplier can enter the product name in the search field. Then, they can either press the Enter key on their keyboard or click the 'Search' button on the interface to initiate the search.

```

77      <!-- Product List -->
78      <ul class="list-group">
79          <% if (products && products.length > 0) { %> <%
80              products.forEach(function(product) { %>
81
82                  <li
83                      class="list-group-item d-flex align-items-center"
84                      data-bs-toggle="modal"
85                      data-bs-target="#manageProductModal<%= product.productID %>">
86
87                      <div class="col-auto me-3">
88                          
94                      </div>
95                      <div class="col">
96                          <h5 class="mb-0"><%= product.productName %></h5>
97                          <p class="mb-0 text-muted">
98                              <%= product.productDescription.length > 70 ?
99                                  product.productDescription.substring(0, 70) + '...' :
100                                  product.productDescription %>
101                          </p>
102                      </div>
103                      <a
104                          href="#"
105                          class="stretched-link"
106                      ></a>
107                      <div class="row justify-content-end">
108                          <div class="col-auto text-center"></div>
109                          <span>RM <%= product.productPrice %> / each</span>
110                      </div>
111                  </li>
112          <% } ); %> <% } else { %>
113              <li class="list-group-item">No products found.</li>
114          <% } %>
115      </ul>
116      <!-- End of Product List -->
```

Figure 6. 143 Product List Section in 'supplierhome.ejs'

The product list section is implemented in the ‘supplierhome.ejs’ file. The mechanism to prompt the user with "No products found" has also been implemented, contributing to an intuitive user experience.

6.7 Advanced-Level Functionalities Implementation

- The Supply Chain Management System shall have the integration with the blockchain.

Figure 6. 144 Advanced-Level Functional Requirements

The advanced-level scope focuses on integrating blockchain with WoodChain. This system utilizes blockchain technology for recording the order and its order status. It means that every action related to order will be stored to the blockchain, such as functions ‘place order’ and ‘update order status’.

As the implementation of smart contract, and configuration of blockchain environment have been discussed in previous part. This section focuses on discussing the how the functions in ‘blockchain.js’.

```
49 module.exports = {
50   web3, // Export web3 instance
51
52   /**
53    * @desc Places an order on the blockchain.
54    * @param {Object} orderData - The order data to be placed.
55    * @param {string} orderData.userID - The Ethereum address of the user placing the order.
56    * @param {string} orderData.supplierID - The ID of the supplier.
57    * @param {string} orderData.deliveryDate - The delivery date of the order.
58    * @param {number} orderData.totalPrice - The total price of the order.
59    * @param {Array<Object>} orderData.orderDetails - The details of the order, including products.
60    * @returns {Promise<void>}
61    * @throws {Error} - Throws an error if there is an issue placing the order on the blockchain.
62   */
63   placeOrder: async ({ userID, supplierID, deliveryDate, totalPrice, orderDetails }) => { ... },
64
65   /**
66    * @desc Updates the status of an order on the blockchain.
67    * @param {Object} statusData - The data for updating the order status.
68    * @param {string} statusData.orderID - The ID of the order to update.
69    * @param {string} statusData.status - The new status of the order ('Confirmed' or 'Pending').
70    * @param {string} statusData.userID - The Ethereum address of the user updating the status.
71    * @returns {Promise<void>}
72    * @throws {Error} - Throws an error if there is an issue updating the order status on the blockchain.
73   */
74   updateOrderStatus: async ({ orderID, status, userID }) => { ... },
75
76   /**
77    * @desc Fetches an order from the blockchain by its ID.
78    * @param {string} orderID - The ID of the order to fetch.
79    * @returns {Promise<Object>} - The order details.
80    * @throws {Error} - Throws an error if there is an issue fetching the order from the blockchain.
81   */
82   getOrder: async (orderID) => { ... },
83
84   /**
85    * @desc Fetches the details of an order from the blockchain by its ID.
86    * @param {string} orderID - The ID of the order to fetch details for.
87    * @returns {Promise<Object>} - The order details.
88    * @throws {Error} - Throws an error if there is an issue fetching the order details from the blockchain.
89   */
90   getOrderDetails: async (orderID) => { ... },
91
92 };
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156 };
```

Figure 6. 145 Exported Functions in ‘blockchain.js’

These functions are implemented by first importing the necessary modules and initializing the contracts. They have been discussed in the previous section, which covers the blockchain configuration.

The functions, ‘placeOrder’, ‘updateOrderStatus’, ‘getOrder’, ‘getOrderDetails’, are used to facilitate interaction with the blockchain for the WoodChain.

```

52  /**
53  * @desc Places an order on the blockchain.
54  * @param {Object} orderData - The order data to be placed.
55  * @param {string} orderData.userID - The Ethereum address of the user placing the order.
56  * @param {string} orderData.supplierID - The ID of the supplier.
57  * @param {string} orderData.deliveryDate - The delivery date of the order.
58  * @param {number} orderData.totalPrice - The total price of the order.
59  * @param {Array<Object>} orderData.orderDetails - The details of the order, including products.
60  * @returns {Promise<void>}
61  * @throws {Error} - Throws an error if there is an issue placing the order on the blockchain.
62  */
63 placeOrder: async ({ userID, supplierID, deliveryDate, totalPrice, orderDetails }) => {
64   try {
65     const deliveryDateTimestamp = Math.floor(new Date(deliveryDate).getTime() / 1000);
66     const accounts = await web3.eth.getAccounts();
67     const instance = await initializedContract;
68
69     const totalPriceCents = Math.round(totalPrice * 100);
70     const formattedOrderDetails = orderDetails.map(detail => ({
71       orderID: detail.orderID,
72       productID: detail.productID,
73       productName: detail.productName,
74       productDescription: detail.productDescription,
75       quantity: detail.quantity,
76       price: Math.round(detail.price * 100) // Convert price to cents
77     }));
78
79     console.log("Placing order with details:", {
80       userID,
81       supplierID,
82       deliveryDateTimestamp,
83       totalPriceCents,
84       formattedOrderDetails
85     });
86
87     await instance.methods.placeOrder(
88       supplierID,
89       deliveryDateTimestamp,
90       totalPriceCents,
91       formattedOrderDetails
92     ).send({ from: userID, gas: 8000000 });
93   } catch (error) {
94     console.error('Error placing order on blockchain:', error);
95     throw error;
96   }
97 },

```

Figure 6. 146 Function for Placing Order in ‘blockchain.js’

This function is used to place an order on the blockchain. It handles the conversion of order data into the appropriate format and sends it to the blockchain using a smart contract method.

It takes in the ‘user ID’, ‘supplier ID’, ‘deliveryDate’, ‘totalPrice’, and ‘orderDetails’, converting the ‘orderDetails’ to a timestamp and the ‘totalPrice’ to cents. Since, it is the appropriate practice to ensure the blockchain can record the data well.

The function retrieves Ethereum accounts and formats each item in the order details with prices in cents. After logging the order details, it sends a transaction to the blockchain’s ‘placeOrder’ method using the user’s Ethereum address, with a specified gas limit for the transaction. This function is then exported and utilized in ‘recordOrder’ function within ‘userController.js’.

```

99  /**
100 * @desc Updates the status of an order on the blockchain.
101 * @param {Object} statusData - The data for updating the order status.
102 * @param {string} statusData.orderID - The ID of the order to update.
103 * @param {string} statusData.status - The new status of the order ('Confirmed' or 'Pending').
104 * @param {string} statusData.userID - The Ethereum address of the user updating the status.
105 * @returns {Promise<void>}
106 * @throws {Error} - Throws an error if there is an issue updating the order status on the blockchain.
107 */
108 updateOrderStatus: async ({ orderID, status, userID }) => {
109     try {
110         console.log(`Updating blockchain order status: orderID=${orderID}, status=${status}, userID=${userID}`);
111         if (!orderID || typeof status === 'undefined' || !userID) {
112             throw new Error('Missing or invalid parameters');
113         }
114         const instance = await initializedContract;
115         const statusEnum = status === 'Confirmed' ? 1 : 0;
116         await instance.methods.updateOrderStatus(orderID, statusEnum).send({ from: userID, gas: 8000000 });
117     } catch (error) {
118         console.error('Error updating order status on blockchain:', error);
119         throw error;
120     }
121 },

```

Figure 6. 147 Function for Updating Order Status in ‘blockchain.js’

The ‘updateOrderStatus’ function in ‘blockchain.js’ is designed to update the status of an order on the blockchain. It takes in the ‘orderID’, ‘status’, and ‘userID’ as parameters. The function first validates the input parameters, ensuring none are missing or invalid. It then retrieves an instance of the initialized contract and converts the status to an appropriate enum value (1 for "Confirmed" and 0 for "Pending"). This function is being used in function ‘updateOrderStatus’ within ‘supplierController.js’.

```

124 /**
125 * @desc Fetches an order from the blockchain by its ID.
126 * @param {string} orderID - The ID of the order to fetch.
127 * @returns {Promise<Object>} - The order details.
128 * @throws {Error} - Throws an error if there is an issue fetching the order from the blockchain.
129 */
130 getOrder: async (orderID) => {
131     try {
132         const instance = await initializedContract;
133         return await instance.methods.getOrder(orderID).call();
134     } catch (error) {
135         console.error('Error fetching order from blockchain:', error);
136         throw error;
137     }
138 },
139
140 /**
141 * @desc Fetches the details of an order from the blockchain by its ID.
142 * @param {string} orderID - The ID of the order to fetch details for.
143 * @returns {Promise<Object>} - The order details.
144 * @throws {Error} - Throws an error if there is an issue fetching the order details from the blockchain.
145 */
146 getOrderDetails: async (orderID) => {
147     try {
148         const instance = await initializedContract;
149         return await instance.methods.getOrderDetails(orderID).call();
150     } catch (error) {
151         console.error('Error fetching order details from blockchain:', error);
152         throw error;
153     }
154 },
155
156 };

```

Figure 6. 148 Functions for Fetching Orders and Order Details in ‘blockchain.js’

They are responsible for fetching orders and order details from the blockchain.

Chapter 7 Testing

This chapter explores the testing progress of the Blockchain Supply Chain Management System. It offers a detailed analysis of the testing procedures, including the specific test cases and the results obtained from the tests.

7.1 Overview of Software Testing

Software testing refers to the procedure of evaluating and verifying whether a system meets the programmer's and final user's requirements (Kasauli et al., 2021). This is also the fourth step of procedure in the research's purposed SDLC. This step significantly contributes to the error, fault, and missing requirements identification and correction. Therefore, the performance, usability, quality, and reliability of the system will also be assessed.

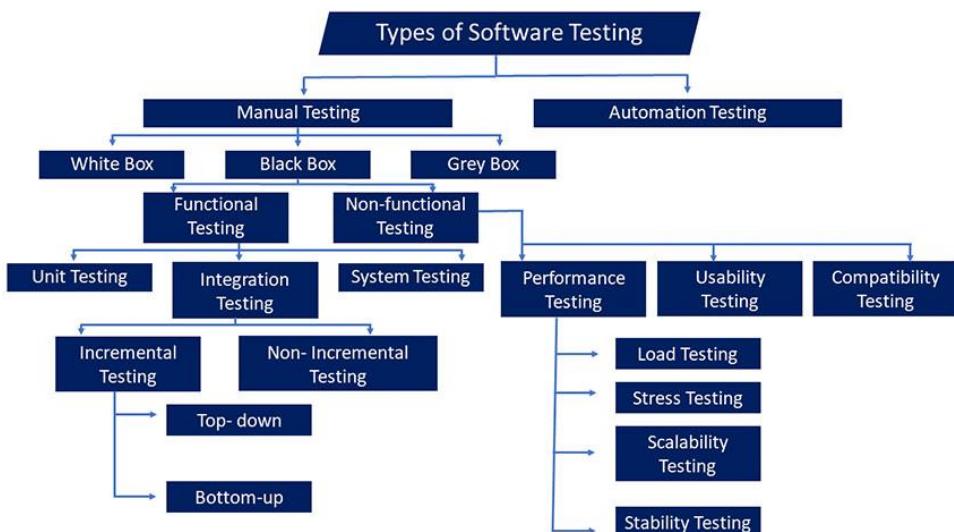


Figure 7. 1 Types of Software Testing (EduBridge, 2023)

The image provided by EduBridge demonstrates that the software testing can be performed manually or automatically. Manual testing encompasses white box, black box, and grey box testing methods. Additionally, testing can be categorized into different levels, namely unit testing, integration testing, and system testing.

7.2 White Box Testing

According to Akhtar's article, white box testing is also known as structural, code-based and glass box testing (2023). This technique evaluates the internal structural design and implementation of a system by granting testers access to the full source code and design documents. It involves programmers who act as testers, creating test cases and debugging the system. This method allows for the identification and resolution of issues that may not be detectable through other testing approaches.

- Advantages:

- Comprehensive coverage
 - Early bug detection
 - System performance optimization
 - Documentation
- Disadvantages
 - High skill requirement
 - Resource intensive

7.3 Black Box Testing

Ashtari stated that black box testing is contrasted to white box testing where the testers do not peer into system internal structures and working (2022). Therefore, the testers evaluate the system based on the expectation that comes with intuitiveness. This approach is effective for evaluating all critical subsystems, including the user interface, user experience, web servers, APIs, and more.

- Advantages:
 - No skill requirement
 - Unbiased results
 - User centric
- Disadvantages:
 - Limited coverage
 - Potential code redundancy

7.4 Grey Box Testing

Grey box testing is a software testing method that serves as the combination of black box and white box testing methodologies (Collins, 2023). In this method, the testers have partial knowledge of the internal system. So, the accessibility of documentation and source code will not be as complete as white box testing. However, this allows the tester to evaluate the system from both the final user's and programmer's perspective.

- Advantages:
 - Reduce code redundancy
 - Efficient testing
- Disadvantages:
 - Moderate skill requirements
 - Potential for oversights

7.5 Testing Levels

7.5.1 Unit Testing

Unit testing is the process of software testing which focuses on testing individual units or components to ensure the individual unit can function correctly (Smartbear, n.d.). This unit can be as small as a function, method, procedure, and so on. This normally ensures that the logic, functionality and behaviour of the testing unit is correct.

7.5.2 Integration Testing

Integration testing focuses on testing the interaction and integration between different components or modules of a software system (Schmitt, 2021). So, the defects which occur only when the integrated components work together, will be detected and solved by performing integration testing.

7.5.3 System Testing

System testing represents a high-level testing phase which tests the complete and integrated software product to ensure it meets requirement (Yasar & Black, 2023). Sometimes, the system test will happen in the user scenarios for ensuring that the system can work properly under realistic conditions.

7.6 Test Case

7.6.1 Sign Up

Test Case ID	TC-01-001		
Test Case Name	Validate User Type		
Related Feature ID	F001		
Objective	1. To test valid user type, “User” and “Supplier” from dropdown menu. 2. To test invalid user type, “Please Select User Type” from dropdown menu.		
Input	Expected Result	Actual Result	Remark
1 User type = “User”	The system accepts the input	The system accepts the input	Pass
2 User type = “Supplier”	The system accepts the input	The system accepts the input	Pass
3 User type = “Please Select User Type”	The system displays the error message “Please select a user type before signing up”	The system displays the error message “Please select a user type before signing up”	Pass

Table 7. 1 Test Case – Validate User Type

Test Case ID	TC-01-002		
Test Case Name	Validate Company Address		
Related Feature ID	F001		
Objective	1. To test invalid company address, company address = 0 character and company address >255 characters. 2. To test valid company address, 0 character < company address < 256 characters. 3. To test invalid company address consisting solely of whitespace characters.		
Input	Expected Result	Actual Result	Remark
1 Company address = 0 character	The system displays the message “Please fill out this field”	The system displays the message “Please fill out this field”	Pass
2 Company address = 1 character	The system accepts the input	The system accepts the input	Pass

3	Company address = 255 characters	The system accepts the input	The system accepts the input	Pass
4	Company address =256 characters	The system stops accepting user input	The system stops accepting user input	Pass
5	Company address = “ ”	The system displays the message “Please match the requested format”	The system displays the message “Please match the requested format”	Pass

Table 7. 2 Test Case – Validate Company Address

Test Case ID		TC-01-003		
Test Case Name		Validate Company Name		
Related Feature ID		F001		
Objective		1. To test invalid company name, company name = 0 character and company name >255 characters. 2. To test valid company name, 0 character < company name < 256 characters. 3. To test invalid company name consisting solely of whitespace characters.		
Input		Expected Result	Actual Result	Remark
1	Company name = 0 character	The system displays the message “Please fill out this field”	The system displays the message “Please fill out this field”	Pass
2	Company name = 1 character	The system accepts the input	The system accepts the input	Pass
3	Company name = 255 characters	The system accepts the input	The system accepts the input	Pass
4	Company name = 256 characters	The system stops accepting user input	The system stops accepting user input	Pass

5	Company name = “ ”	The system displays the message “Please match the requested format”	The system displays the message “Please match the requested format”	Pass
---	--------------------	---	---	------

Table 7. 3 Test Case – Validate Company Name

Test Case ID		TC-01-004		
Test Case Name		Validate Email		
Related Feature ID		F001		
Objective		1. To test invalid Email, Email = 0 character and Email >255 characters. 2. To test valid Email, 0 character < Email < 256 characters. 3. To test invalid Email consisting solely of whitespace characters. 4. To test invalid Email for missing or incorrectly positioned “@” or “.”. 5. To test invalid Email with registered Email.		
Input		Expected Result	Actual Result	Remark
1	Email = 0 character	The system displays the message “Please fill out this field”	The system displays the message “Please fill out this field”	Pass
2	Email = 255 characters	The system accepts the input	The system accepts the input	Pass
3	Email = 256 characters	The system stops accepting user input	The system stops accepting user input	Pass
4	Email = “ ”	The system displays the message “Please fill out this field”	The system displays the message “Please fill out this field”	Pass
5	Email = “ abc@abc.com”	The system accepts the input	The system accepts the input	Pass
6	Email = “@.com”	The system displays the message “Please enter a part followed by ‘@.’ ‘@.’ is incomplete”	The system displays the message “Please enter a part followed by ‘@.’ ‘@.’ is incomplete”	Pass
7	Email = “abc”	The system displays the message “Please include an ‘@’ in the email address. ‘abc’ is missing an ‘@’”	The system displays the message “Please include an ‘@’ in the email address. ‘abc’ is missing an ‘@’”	Pass

8	Email = “abc@”	The system displays the message “Please enter a part following ‘@’. ‘abc@’ is incomplete”	The system displays the message “Please enter a part following ‘@’. ‘abc@’ is incomplete”	Pass
9	Email registered account email	The system displays the error message “User already exist with the provided email”	The system displays the error message “User already exist with the provided email”	Pass

Table 7. 4 Test Case – Validate Email

Test Case ID		TC-01-005		
Test Case Name		Validate Password		
Related Feature ID		F001		
Objective		1. To test invalid password, password < 8 characters and password >255 characters. 2. To test invalid password, password = 0 character. 3. To test valid company name, 7 characters < password < 256 characters. 4. To test invalid password consisting solely of whitespace characters.		
Input		Expected Result	Actual Result	Remark
1	Password = 0	The system displays the message “Please fill out this field”	The system displays the message “Please fill out this field”	Pass
2	Password = 7 characters	The system displays the message “Please lengthen this text to 8 characters or more (you currently using 7 characters)”	The system displays the message “Please lengthen this text to 8 characters or more (you currently using x characters)”	Pass
3	Password = 8 characters	The system accepts the input	The system accepts the input	Pass
4	Password = 255 characters	The system accepts the input	The system accepts the input	Pass
5	Password = 256 characters	The system stops accepting user input	The system stops accepting user input	Pass

6	Password = “ ”	The system displays the message “Please match the requested format”	The system displays the message “Please match the requested format”	Pass
---	----------------	---	---	------

Table 7. 5 Test Case – Validate Password

7.6.2 Log In

Test Case ID	TC-02-001		
Test Case Name	Validate Email		
Related Feature ID	F002		
Objective	<ol style="list-style-type: none"> 1. To test invalid Email, Email = 0 character and Email >255 characters. 2. To test valid Email, 0 character < Email < 256 characters. 3. To test invalid Email consisting solely of whitespace characters. 4. To test invalid Email for missing or incorrectly positioned “@” or “.”. 5. To test invalid Email with registered Email. 		
Input	Expected Result	Actual Result	Remark
1 Email = 0 character	The system displays the message “Please fill out this field”	The system displays the message “Please fill out this field”	Pass
2 Email = 255 characters	The system accepts the input	The system accepts the input	Pass
3 Email = 256 characters	The system stops accepting user input	The system stops accepting user input	Pass
4 Email = “ ”	The system displays the message “Please fill out this field”	The system displays the message “Please fill out this field”	Pass
5 Email = “ abc@abc.com”	The system accepts the input	The system accepts the input	Pass
6 Email = “@.com”	The system displays the message “Please enter a part followed by ‘@.’ ‘@.’ is incomplete”	The system displays the message “Please enter a part followed by ‘@.’ ‘@.’ is incomplete”	Pass
7 Email = “abc”	The system displays the message “Please include an ‘@’ in the email address. ‘abc’ is missing an ‘@’”	The system displays the message “Please include an ‘@’ in the email address. ‘abc’ is missing an ‘@’”	Pass

8	Email = “abc@”	The system displays the message “Please enter a part following ‘@’. ‘abc@’ is incomplete”	The system displays the message “Please enter a part following ‘@’. ‘abc@’ is incomplete”	Pass
9	Email unregistered account email	= The system displays the error message “Invalid email or password”	The system displays the error message “Invalid email or password”	Pass

Table 7. 6 Test Case – Validate Email

Test Case ID		TC-02-002		
Test Case Name		Validate Password		
Related Feature ID		F002		
Objective		1. To test invalid Email consisting solely of whitespace characters. 2. To test invalid password, password = 0 character.		
Input		Expected Result	Actual Result	Remark
1	Password registered account password	= The system directs user or supplier to the respective homepage	The system directs user or supplier to the respective homepage	Pass
2	Password wrong account password	= The system displays the message “Invalid email or password”	The system displays the message “Invalid email or password”	Pass
1	Password = “ ”	= The system displays the message “Invalid email or password”	The system displays the message “Invalid email or password”	Pass
2	Password = 0 character	= The system displays the message “Please fill out this field”	The system displays the message “Please fill out this field”	Pass

Table 7. 7 Test Case – Validate Password

Test Case ID		TC-02-002		
Test Case Name		Validate Log In Function		
Related Feature ID		F002		
Objective		1. To test whether the system can effectively perform the authentication process with the correct email and password.		

		2. To test whether the system can effectively perform the authentication process with the incorrect email or password.		
Input		Expected Result	Actual Result	Remark
1	Email and password are both correct	The system directs user or supplier to the respective homepage	The system directs user or supplier to the respective homepage	Pass
2	Email and password are both incorrect	The system displays the message “Invalid email or password”	The system displays the message “Invalid email or password”	Pass
2	Email or password is incorrect	The system displays the message “Invalid email or password”	The system displays the message “Invalid email or password”	Pass

Table 7. 8 Test Case – Validate Log In Function

7.6.3 Select Supplier

Test Case ID	TC-03-001			
Test Case Name	Access Control Test for Select Supplier page			
Related Feature ID	F003			
Objective	1. To test unauthorized access to Select Supplier page via URL duplication. 2. To test authorized access to Select Supplier page via URL duplication.			
Input	Expected Result	Actual Result		Remark
1 User copies the URL into the web browser without being logged in	The system directs user to Log In page with error message “Please Login first”	The system directs user to Log In page with error message “Please Login first”		Pass
2 User copies the URL into the web browser after being logged in	The system directs user to Select Supplier page	The system directs user to Select Supplier page		Pass

Table 7. 9 Test Case – Access Control Test

Test Case ID	TC-03-002
---------------------	-----------

Test Case Name		Validate display of all registered suppliers		
Related Feature ID		F003		
Objective		1. To test Select Supplier page accurately displays all registered suppliers		
Input		Expected Result	Actual Result	Remark
1	User is directed to Select Supplier page	The system displays all the suppliers in the list	The system displays all the suppliers in the list	Pass

Table 7. 10 Test Case – Validate Display of All Registered Suppliers

Test Case ID		TC-03-003		
Test Case Name		Validate user capability to select suppliers and display associated products		
Related Feature ID		F003		
Objective		1. To test whether the user can select a supplier and display the products associated with that supplier.		
Input		Expected Result	Actual Result	Remark
1	User selects a supplier	The system directs user to Select Product page with products associated with selected supplier	The system directs user to Select Product page with products associated with selected supplier	Pass

Table 7. 11 Test Case – Validate User Capability to Select Suppliers and Display Associated Products

7.6.4 Select Product

Test Case ID		TC-04-001		
Test Case Name		Access Control Test for Select Product page		
Related Feature ID		F004		
Objective		1. To test unauthorized access to Select Product page via URL duplication. 2. To test authorized access to Select Product page via URL duplication.		
Input		Expected Result	Actual Result	Remark
1	User copies the URL into the web browser without being logged in	The system directs user to Log In page with error message “Please Login first”	The system directs user to Log In page with error message “Please Login first”	Pass

2	User copies the URL into the web browser after being logged in	The system directs user to Select Product page	The system directs user to Select Product page	Pass
---	--	--	--	------

Table 7. 12 Test Case – Access Control Test

Test Case ID		TC-04-002		
Test Case Name		Validate Select Product and Order Summary		
Related Feature ID		F004		
Objective		1. To test whether the order summary can record the selected product when the user selects product in the product list. 2. To test whether the order summary can retain selected product after logging out. 3. To test whether the delete icon in order summary can be used.		
Input		Expected Result	Actual Result	Remark
1	User selects the product by selecting quantity	The order summary displays the selected product's name, quantity, and price	The order summary displays the selected product's name, quantity, and price	Pass
2	User selects the product by selecting quantity then log out	The order summary retains the selected product's name, quantity, and price	The order summary retains the selected product's name, quantity, and price	Pass
3	User selects the delete icon associated with the product in order summary	The product quantity in product list becomes 0 and is subsequently removed from order summary	The product quantity in product list becomes 0 and is subsequently removed from order summary	Pass

Table 7. 13 Test Case – Validate Select Product and Order Summary

Test Case ID		TC-04-003
Test Case Name		Validate Proceed to Checkout Function

Related Feature ID	F004		
Objective	1. To test whether the user can proceed to checkout with or without selecting a product.		
Input	Expected Result	Actual Result	Remark
1 User selects “Proceed to Checkout” button with selected product	The system successfully directs user to Place Order page, correctly displaying supplier information, order date, and a summary of the order	The system successfully directs user to Place Order page, correctly displaying supplier information, order date, and a summary of the order	Pass
2 User selects “Proceed to Checkout” button without selected product	The system displays the message “No products selected. Please add products to your order before proceeding to check out.”	The system displays the message “No products selected. Please add products to your order before proceeding to check out.”	Pass

Table 7. 14 Test Case – Validate Proceed to Checkout Function

7.6.5 Place Order

Test Case ID	TC-05-001		
Test Case Name	Access Control Test for Place Order page		
Related Feature ID	F005		
Objective	1. To test unauthorized access to Place Order page via URL duplication. 2. To test authorized access to Place Order page via URL duplication.		
Input	Expected Result	Actual Result	Remark
1 User copies the URL into the web browser without being logged in	The system directs user to Log In page with error message “Please Login first”	The system directs user to Log In page with error message “Please Login first”	Pass
2 User copies the URL into the web browser after being logged in	The system directs user to Place Order page	The system directs user to Place Order page	Pass

Table 7. 15 Test Case – Validate Access Control

Test Case ID	TC-05-002		
Test Case Name	Validate Delivery Date		
Related Feature ID	F005		
Objective	<ol style="list-style-type: none"> 1. To test whether the user can select a delivery date which has already passed. 2. To test whether the user can enter the delivery date manually. 		
Input	Expected Result	Actual Result	Remark
1 User selects the delivery date field	The system displays the date picker and make the date that has already passed is not selectable	The system displays the date picker and make the date that has already passed is not selectable	Pass
2 User selects the delivery date field	The system does not allow user to enter the delivery date manually	The system does not allow user to enter the delivery date manually	Pass

Table 7. 16 Test Case – Validate Delivery Date

Test Case ID	TC-05-003		
Test Case Name	Validate Place Order Function		
Related Feature ID	F005		
Objective	<ol style="list-style-type: none"> 1. To test whether user can place order before selecting the delivery date. 		
Input	Expected Result	Actual Result	Remark
1 User selects “Place Order” button and select “confirm” button in confirmation modal with selected delivery date	The system displays directs to Manage Order page and displays the message “Order has been placed successfully”	The system directs to Manage Order page and displays the message “Order has been placed successfully”	Pass
2 User selects “Proceed to Checkout” button without	The system displays the message “Please select the delivery date first”	The system displays the message “Please select the delivery date first”	Pass

	selected delivery date			
--	------------------------	--	--	--

Table 7. 17 Test Case – Validate Place Order Function

7.6.6 Check Order Status

Test Case ID		TC-06-001		
Test Case Name		Access Control Test for Manage Order page.		
Related Feature ID		F006		
Objective		1. To test unauthorized access to Manage Order page via URL duplication. 2. To test authorized access to Manage Order page via URL duplication.		
Input		Expected Result	Actual Result	Remark
1	User or supplier copies the URL into the web browser without being logged in	The system directs user to Log In page with error message “Please Login first”	The system directs user to Log In page with error message “Please Login first”	Pass
2	User or supplier copies the URL into the web browser after being logged in	The system directs user to Manage Order page	The system directs user to Manage Order page	Pass

Table 7. 18 Test Case – Access Control Test

Test Case ID		TC-06-002		
Test Case Name		Validate display of all orders		
Related Feature ID		F006		
Objective		1. To test Manage Order page accurately displays all order associated with the logged-in user or supplier.		
Input		Expected Result	Actual Result	Remark
1	User is directed to Manage Order page	The system displays all the orders and each order status correctly.	The system displays all the orders and each order status correctly.	Pass

Table 7. 19 Test Case – Validate Display of All Orders

7.6.7 Check Invoice

Test Case ID	TC-07-001		
Test Case Name	Access Control Test for Check Invoice page		
Related Feature ID	F007		
Objective	1. To test unauthorized access to PDF invoice via URL duplication. 2. To test authorized access to PDF invoice via URL duplication.		
Input	Expected Result	Actual Result	Remark
1 User or supplier copies the URL into the web browser without being logged in	The system directs user to Log In page with error message “Please Login first”.	The system directs user to Log In page with error message “Please Login first”.	Pass
2 User or supplier copies the URL into the web browser after being logged in	The system directs user to PDF invoice.	The system directs user to Select Product page.	Pass

Table 7. 20 Test Case – Access Control Test

Test Case ID	TC-07-002		
Test Case Name	Validate Check Invoice Function		
Related Feature ID	F007		
Objective	1. To test whether the invoice can be generated for the order which has not been confirmed. 2. To test whether the system can display waiting message when invoice is being generated.		
Input	Expected Result	Actual Result	Remark
1 User or supplier selects an order which has not been confirmed	The system displays modal which contains order information.	The system displays modal which contains order information.	Pass
2 User or supplier selects an order	The system displays the message “Invoice is generating, please	The system displays the message “Invoice is generating, please	Pass

	which has been confirmed	wait..” and directs user to a new tab with the PDF invoice	wait..” and directs user to a new tab with the PDF invoice	
--	--------------------------	--	--	--

Table 7. 21 Test Case – Validate Check Invoice Function

7.6.8 Confirm Order

Test Case ID		TC-08-001		
Test Case Name		Validate Confirm Order Function		
Related Feature ID		F008		
Objective		1. To test whether the supplier can confirm the order successfully.		
Input		Expected Result	Actual Result	Remark
1	Supplier selects “Confirm Order” button and “Confirm” button in confirmation modal	The system displays the message “Order status updated successfully!” and displays updated order list.	The system displays the message “Order status updated successfully!” and displays updated order list.	Pass

Table 7. 22 Test Case – Validate Confirm Order Function

7.6.9 Edit Product

Test Case ID		TC-09-001		
Test Case Name		Access Control Test for Manage Product page		
Related Feature ID		F009		
Objective		1. To test unauthorized access to Manage Product page via URL duplication. 2. To test authorized access to Manage Product page via URL duplication.		
Input		Expected Result	Actual Result	Remark
1	Supplier copies the URL into the web browser without being logged in	The system directs supplier to Log In page with error message “Please Login first”	The system directs supplier to Log In page with error message “Please Login first”	Pass
2	Supplier copies the URL into	The system directs supplier to Manage Product page	The system directs supplier to Manage Product page	Pass

	the web browser after being logged in			
--	---------------------------------------	--	--	--

Table 7. 23 Test Case – Access Control Test

Test Case ID		TC-09-002		
Test Case Name		Validate display of all associated products		
Related Feature ID		F009		
Objective		1. To test Manage Product page accurately displays all associated products		
Input		Expected Result	Actual Result	Remark
1	Supplier is directed to Manage Product page	The system displays all the products in the list	The system displays all the products in the list	Pass

Table 7. 24 Test Case – Validate Display of All Associated Products

Test Case ID		TC-09-003		
Test Case Name		Validate Product Name		
Related Feature ID		F009		
Objective		1. To test invalid product name, product name = 0 character and product name >255 characters. 2. To test valid product name, 0 character < product name < 256 characters. 3. To test invalid product name consisting solely of whitespace characters.		
Input		Expected Result	Actual Result	Remark
1	Product name = 0 character	The system displays the message “Please fill out this field”	The system displays the message “Please fill out this field”	Pass
2	Product name = 1 character	The system accepts the input	The system accepts the input	Pass
3	Product name = 255 characters	The system accepts the input	The system accepts the input	Pass
4	Product name = 256 characters	The system stops accepting user input	The system stops accepting user input	Pass

5	Product name = “ ”	The system displays the message “Please match the requested format”	The system displays the message “Please match the requested format”	Pass
---	--------------------	---	---	------

Table 7. 25 Test Case – Validate Product Name

Test Case ID		TC-09-004		
Test Case Name		Validate Product Description		
Related Feature ID		F009		
Objective		1. To test invalid product description, product description = 0 character and product description >300 characters. 2. To test valid product name, 0 character < product name < 301 characters. 3. To test invalid product name consisting solely of whitespace characters.		
Input	Expected Result	Actual Result	Remark	
1 Product description = 0 character	The system displays the message “Product Description cannot be empty or only whitespace”	The system displays the message “Product Description cannot be empty or only whitespace”	Pass	
2 Product description = 1 character	The system accepts the input	The system accepts the input	Pass	
3 Product description = 300 characters	The system accepts the input	The system accepts the input	Pass	
4 Product description = 301 characters	The system stops accepting user input	The system stops accepting user input	Pass	
5 Product description = “ ”	The system displays the message “Product Description cannot be empty or only whitespace”	The system displays the message “Product Description cannot be empty or only whitespace”	Pass	

Table 7. 26 Test Case – Validate Product Description

Test Case ID		TC-09-005		
Test Case Name		Validate Product Price		
Related Feature ID		F009		

Objective		<ol style="list-style-type: none"> 1. To test invalid product price, product price = 0 digit and product price > 6 digits. 2. To test invalid product price, product price which is not exactly equal to 2 decimal places. 3. To test valid product price, 0 digit < product price < 7 digits. 4. To test valid product price, product price = 0 decimal places and product price = 2 decimal places. 5. To test invalid product price consisting solely of whitespace characters. 		
Input		Expected Result	Actual Result	Remark
1	Product price = 0 digit	The system displays the message “Please fill out this field”	The system displays the message “Please fill out this field”	Pass
2	Product price = 7 digits	The system displays the message “Please match the requested format”	The system displays the message “Please match the requested format”	Pass
3	Product price = 3 decimal places	The system displays the message “Please match the requested format”	The system displays the message “Please match the requested format”	Pass
4	Product price = 1 decimal places	The system displays the message “Please match the requested format”	The system displays the message “Please match the requested format”	Pass
5	Product price = 1digit	The system accepts the input	The system accepts the input	Pass
6	Product price = 6 digits	The system accepts the input	The system accepts the input	Pass
7	Product price = 2 decimal places	The system accepts the input	The system accepts the input	Pass
8	Product price = 0 decimal places	The system accepts the input	The system accepts the input	Pass
9	Product price = “ ”	The system displays the message “Please match the requested format”	The system displays the message “Please match the requested format”	Pass

Table 7. 27 Test Case – Validate Product Price

Test Case ID	TC-09-006
---------------------	-----------

Test Case Name	Validate Product Photo		
Related Feature ID	F009		
Objective	1. To test whether supplier can select the file formats other than JPG, PNG, JPEG.		
Input	Expected Result	Actual Result	Remark
1 Supplier selects choose file field	The system opens file explorer and only the file with the format JPG, PNG or JPEG is selectable.	The system opens file explorer and only the file with the format JPG, PNG or JPEG is selectable.	Pass

Table 7. 28 Test Case – Validate Product Photo

Test Case ID	TC-09-007		
Test Case Name	Validate Edit Product Information Function		
Related Feature ID	F009		
Objective	1. To test whether the user can effectively update product information or delete product		
Input	Expected Result	Actual Result	Remark
1 User selects “Update Product” button and “Confirm” button in confirmation modal	The system successfully updates product information and displays updated product list	The system successfully updates product information and displays updated product list	Pass
2 User selects “Delete Product” button and “Delete” button in confirmation modal	The system successfully deletes product and displays updated product list	The system successfully deletes product and displays updated product list	Pass

Table 7. 29 Test Case – Validate Edit Product Information Function

7.6.10 Add Product

Test Case ID	TC-10-001		
Test Case Name	Validate Product Name		
Related Feature ID	F010		
Objective	<ol style="list-style-type: none"> 1. To test invalid product name, product name = 0 character and product name >255 characters. 2. To test valid product name, 0 character < product name < 256 characters. 3. To test invalid product name consisting solely of whitespace characters. 		
Input	Expected Result	Actual Result	Remark
1 Product name = 0 character	The system displays the message “Please fill out this field”	The system displays the message “Please fill out this field”	Pass
2 Product name = 1 character	The system accepts the input	The system accepts the input	Pass
3 Product name = 255 characters	The system accepts the input	The system accepts the input	Pass
4 Product name = 256 characters	The system stops accepting user input	The system stops accepting user input	Pass
5 Product name = “ ”	The system displays the message “Please match the requested format”	The system displays the message “Please match the requested format”	Pass

Table 7. 30 Test Case – Validate Add Product Function

Test Case ID	TC-10-002		
Test Case Name	Validate Product Description		
Related Feature ID	F010		
Objective	<ol style="list-style-type: none"> 1. To test invalid product description, product description = 0 character and product description >300 characters. 2. To test valid product description, 0 character < product description < 301 characters. 3. To test invalid product description consisting solely of whitespace characters 		
Input	Expected Result	Actual Result	Remark

1	Product description = 0 character	The system displays the message “Product Description cannot be empty or only whitespace”	The system displays the message “Product Description cannot be empty or only whitespace”	Pass
2	Product description = 1 character	The system accepts the input	The system accepts the input	Pass
3	Product description = 300 characters	The system accepts the input	The system accepts the input	Pass
4	Product description = 301 characters	The system stops accepting user input	The system stops accepting user input	Pass
5	Product description = “ ”	The system displays the message “Product Description cannot be empty or only whitespace”	The system displays the message “Product Description cannot be empty or only whitespace”	Pass

Table 7. 31 Test Case – Validate Product Description

Test Case ID	TC-10-003		
Test Case Name	Validate Product Price		
Related Feature ID	F010		
Objective	1. To test invalid product price, product price = 0 digit and product price > 6 digits. 2. To test invalid product price, product price which is not exactly equal to 2 decimal places 3. To test valid product price, 0 digit < product price < 7 digits. 4. To test valid product price, product price = 0 decimal places and product price = 2 decimal places 5. To test invalid product price consisting solely of whitespace characters		
Input	Expected Result	Actual Result	Remark
1 Product price = 0 digit	The system displays the message “Please fill out this field”	The system displays the message “Please fill out this field”	Pass
2 Product price = 7 digits	The system displays the message “Please match the requested format”	The system displays the message “Please match the requested format”	Pass

3	Product price = 3 decimal places	The system displays the message “Please match the requested format”	The system displays the message “Please match the requested format”	Pass
4	Product price = 1 decimal places	The system displays the message “Please match the requested format”	The system displays the message “Please match the requested format”	Pass
5	Product price = 1digit	The system accepts the input	The system accepts the input	Pass
6	Product price = 6 digits	The system accepts the input	The system accepts the input	Pass
7	Product price = 2 decimal places	The system accepts the input	The system accepts the input	Pass
8	Product price = 0 decimal places	The system accepts the input	The system accepts the input	Pass
9	Product price = “ ”	The system displays the message “Please match the requested format”	The system displays the message “Please match the requested format”	Pass

Table 7. 32 Test Case – Validate Product Price

Test Case ID		TC-10-004		
Test Case Name		Validate Product Photo		
Related Feature ID		F010		
Objective		1. To test whether supplier can select the file formats other than JPG, PNG, JPEG		
Input		Expected Result	Actual Result	Remark
1	Supplier selects choose file field	The system opens file explorer and only the file with the format JPG, PNG or JPEG is selectable.	The system opens file explorer and only the file with the format JPG, PNG or JPEG is selectable.	Pass

Test Case – Validate Product Price

Test Case ID		TC-10-005
Test Case Name		Validate Add Product Function

Related Feature ID		F010		
Objective		1. To test whether the user can effectively add a product to the product list.		
Input		Expected Result	Actual Result	Remark
1	User selects “Add Product” button and “Yee, Add Product” button in confirmation modal	The system successfully adds product to product list and displays updated product list	The system successfully adds product to product list and displays updated product list	Pass

Table 7. 33 Test Case – Validate Add Product Information Function

7.6.11 Check Order Details

Test Case ID		TC-11-001		
Test Case Name		Validate Check Order Details Function		
Related Feature ID		F011		
Objective		1. To test whether the Order Details modal can be displayed for the order which has not been confirmed		
Input		Expected Result	Actual Result	Remark
1	User or supplier selects an order which has not been confirmed	The system displays modal which contains order information.	The system displays modal which contains order information.	Pass

Table 7. 34 Test Case – Validate Check Order Details Function

7.6.12 Add Company Description

Test Case ID		TC-12-001		
Test Case Name		Validate Company Description		
Related Feature ID		F012		
Objective		1. To test invalid company description, company description = 0 character and company description >300 characters. 2. To test valid company description, 0 character < product name < 301 characters. 3. To test invalid company description consisting solely of whitespace characters		

Input	Expected Result	Actual Result	Remark
1 Company description = 0 character	The system displays the message “Company description cannot be empty or only whitespace”	The system displays the message “Company description cannot be empty or only whitespace”	Pass
2 Company description = 1 character	The system accepts the input	The system accepts the input	Pass
3 Company description = 300 characters	The system accepts the input	The system accepts the input	Pass
4 Company description = 301 characters	The system stops accepting user input	The system stops accepting user input	Pass
5 Company description = “ ”	The system displays the message “Product description cannot be empty or only whitespace”	The system displays the message “Product description cannot be empty or only whitespace”	Pass

Table 7. 35 Test Case – Validate Company Description

7.6.13 Upload Profile Photo

Test Case ID	TC-13-001		
Test Case Name	Validate Product Photo		
Related Feature ID	F013		
Objective	1. To test whether user or supplier can select the file formats other than JPG, PNG, JPEG		
Input	Expected Result	Actual Result	Remark
1 User or supplier selects choose file field	The system opens file explorer and only the file with the format JPG, JPEG, PNG or JPEG is selectable.	The system opens file explorer and only the file with the format JPG, JPEG, PNG or JPEG is selectable.	Pass

Table 7. 36 Test Case – Validate Profile Photo

7.6.14 Search Supplier

Test Case ID	TC-14-001		
Test Case Name	Validate Supplier Search by Name		

Related Feature ID	F014		
Objective	<ol style="list-style-type: none"> 1. To test whether user can search supplier by entering the supplier's name 2. To test whether user can search supplier by entering partial names or substrings of supplier's name 3. To test whether the system can display message if no supplier is found 		
Input	Expected Result	Actual Result	Remark
1 User enters supplier's company name	The system displays the corresponding supplier.	The system displays the corresponding supplier.	Pass
2 User enters partial names or substrings of supplier's name	The system displays the corresponding supplier.	The system displays the corresponding supplier.	Pass
3 User enters incorrect supplier's name	The system displays message, indicating that no supplier is found.	The system displays message, indicating that no supplier is found.	Pass

Table 7. 37 Test Case – Validate Supplier Search by Name

7.6.15 Search Order

Test Case ID	TC-15-001		
Test Case Name	Validate Order Search by Name		
Related Feature ID	F015		
Objective	<ol style="list-style-type: none"> 1. To test whether user can enter the supplier's name for searching the order associated with the searched supplier 2. To test whether supplier can enter the customer's name for searching the order associated with the searched customer 3. To test whether user or supplier can search order by entering partial names or substrings of supplier's or customer's name 4. To test whether the system can display message if no order is found 		
Input	Expected Result	Actual Result	Remark
1 User enters supplier's company name	The system displays the corresponding order.	The system displays the corresponding order.	Pass

2	Supplier enters customer's name	The system displays the corresponding order.	The system displays the corresponding order.	Pass
3	User or supplier enters partial names or substrings of supplier's name	The system displays the corresponding order.	The system displays the corresponding order.	Pass
4	User or supplier enters incorrect name	The system displays message, indicating that no order is found.	The system displays message, indicating that no order is found.	Pass

Table 7. 38 Test Case – Validate Order Search By Name

7.6.16 Search Product

Test Case ID		TC-16-001		
Test Case Name		Validate Product Search by Name		
Related Feature ID		F016		
Objective		1. To test whether supplier can search product by entering the product's name 2. To test whether user can search product by entering partial names or substrings of product's name 3. To test whether the system can display if no product is found		
Input		Expected Result	Actual Result	Remark
1	Supplier enters product's name	The system displays the corresponding product.	The system displays the corresponding supplier.	Pass
2	Supplier enters partial names or substrings of product's name	The system displays the corresponding supplier.	The system displays the corresponding supplier.	Pass
3	Supplier enters incorrect product's name	The system displays message, indicating that no product is found.	The system displays message, indicating that no product is found.	Pass

Table 7. 39 Test Case – Validate Product Search by Name

7.6.17 Sign Out

Test Case ID	TC-17-001		
Test Case Name	Validate Sign Out Function		
Related Feature ID	F017		
Objective	<ol style="list-style-type: none"> 1. To test whether the sign-out functionality can effectively log the user or supplier out. 2. To test whether the system can protect the route if user navigate to a previously accessible authenticated page by using browser back button or entering the URL directly. 		
Input	Expected Result	Actual Result	Remark
1 User or supplier selects ‘Sign Out’ button in confirmation modal	The system directs user or supplier to Log In page	The system directs user or supplier to Log In page	Pass
2 User or supplier selects back button or entering the URL in web browser	The system redirects users or suppliers to their previous page or to a specific target page, while preventing them from performing any actions unless they are properly authenticated.	The system redirects users or suppliers to their previous page or to a specific target page, while preventing them from performing any actions unless they are properly authenticated.	Pass

Table 7. 40 Test Case – Validate Sign Out Function

7.7 User Acceptance Testing

7.7.1 UAT 1

User Acceptance Testing					
<u>Blockchain Technology Supply Chain Management System</u>					
WoodChain is a Supply Chain Management System designed to assist the SME customization furniture industry in managing the raw material ordering process. The system offers functionalities for both suppliers and users to streamline their raw material orders and generate invoices. To ensure transparency and immutability of transactions, order information is recorded on both the blockchain and a database. The interface is designed to be as simple as possible to quickly engage new users, adopting an e-commerce platform style for ease of use, even though it is intended for business purposes.					
Tester Name: JT	Position: Secretary		Date: 13 – 07 - 2024		
Name of Category	Score (1 is the lowest, 5 is the highest)				
	1	2	3	4	5
User Friendliness				/	
Interface Design					/
User Feedback				/	
Business Requirement Fulfilment				/	
Comment From Tester	Overall, the system is easy to use. The interface quite nice, just lacking instruction for the available functions, like first time use can't know that the order can be selected to generate the invoice. In terms of the functionality, it covers the essential needs in my daily operation. For example, it can select the supplier for choosing product and place the order. The most important is that the delivery date can be selected for larger flexibility in order process. If possible, the system should have more functions like editing order, address and so on to make it approach to the real-world operation.				
Action Taken by Developer	The comment from JT represents generally positive feedback. To address issues like insufficient instructions, the cursor has been changed to a pointer when hovering over clickable areas, indicating potential functionality. While redesigning the interface to provide clearer and more direct instructions would be ideal, time constraints have limited this to the best available solution for now. JT's suggestions for additional features have also been noted for future enhancements.				

Table 7. 41 User Acceptance Testing

Chapter 8 Critical Evaluation and Conclusion

This chapter provides a comprehensive analysis and evaluation of the project. It covers various aspects, including project management, self-performance, and the challenges faced during the project. Additionally, potential areas for future enhancement will be explored to ensure personal growth and to improve the system based on the lessons learned from this project.

8.1 Process Evaluation

WoodChain, a Supply Chain Management System for the SME customization furniture industry, was inspired by the persistent supply chain issues that hinder profitability in this sector. The scarcity of articles on this topic highlighted an urgent need for research and potential solutions to support businesses in this industry. The problem statement, "The customization furniture industry is still stuck in Industry 2.0," underscored the necessity of advancing to Industry 4.0. This solution aimed to address this problem by modernizing the industry and tackling its current challenges. Moreover, the application of blockchain technology also contributed to this goal.

In short, the project aimed to tackle the supply chain issues in the customization furniture industry by offering a system that replaced the traditional manual methods of managing supply-chain-related business processes. This system was designed to streamline operations, enhance efficiency, and modernize the industry's approach to supply chain management.

As a student studying Enterprise Information Systems, it was important to recognize that elements such as business processes, business requirements, project constraints, interface design, and data analysis were as crucial as the technical implementation. Therefore, these elements were thoroughly analyzed to ensure the final product was usable, easy to use, user-friendly, and reliable. Consequently, considerable time was invested in conducting literature reviews to study business processes, identify potential business requirements, and analyze collected data to validate the proposed system's significance.

After conducting the research on development frameworks, programming languages, interface design, and the system design phase commenced. This phase included the creation of various UML diagrams and interfaces that became the foundation for the actual system implementation.

The implementation phase included the creation of the interface, blockchain, and back-end logic. The business logic gleaned from the literature review played a valuable role in both the front-end and back-end development. For instance, given the constraints of knowledge and time available for learning to use the system, the front-end was designed to be as simple as possible. The back-end logic was implemented based on business processes, incorporating reasonable database manipulations. Examples included generating invoices only when both parties agreed on transactions and implementing product soft delete logic.

Throughout the project, considerable time was spent in meetings with the project supervisor. The supervisor provided supportive feedback and assistance in conducting literature reviews, implementation, system testing, and even documentation.

Eventually, the project successfully developed a supply chain management system with blockchain based on the planned project scope. Despite the existence of some shortcomings which have to be admitted, WoodChain represents a significant step towards satisfying the problem of related industry.

8.3 Challenges

The biggest challenge faced would be the implementation of the system. Since the frequency of dealing with programming language is not much and could even be considered as the rookie of fresher, it posed a significant challenge in working with that many libraries, languages, and frameworks. In addition, the utilization of the MVC system architecture and the configuration of the blockchain presented further difficulties for a beginner.

Challenges arose from the most basic steps, such as setting up the development environment in the IDE, to the actual writing of the code. For instance, the process of writing HTML, CSS, JavaScript, SQL queries, and Solidity was not smooth, particularly with client-side JavaScript demanding considerable effort. Among all tasks, configuring the blockchain and implementing smart contracts were the most challenging aspects.

The project management was deemed the second major challenge in this project. For example, Git was not utilized due to knowledge limitations, necessitating manual duplication of the development project to ensure that erroneous implementations could be rolled back.

The inexperienced system design also brought challenges, particularly with creating UML diagrams used to represent system flows from a conceptual perspective. However, the assistance provided by the project supervisor helped address this challenge to a certain degree. The supervisor's feedback on UML diagrams was instrumental in improving the function and system flow before the implementation began.

Lastly, writing Chapter 6 was relatively complex due to the extensive directory structure. Organizing the content presented a challenge, but with careful planning, the documentation still maintains a good flow.

8.4 Self-Evaluation

Instead of discussing personal strengths and weaknesses only, this section will also delve into personal growth. It will explore how the project contributed to the development of new skills, the enhancement of existing abilities, and the overall improvement of professional and personal competencies.

8.4.1 Enthusiasm for Learning

One of the most significant strengths demonstrated throughout the project was a strong willingness to learn new skills, techniques, technologies, languages, and frameworks. For instance, although PHP and XAMPP had been learned and applied previously, this project required the application of Node.js, Express, EJS, Bootstrap, Sqlite3, Solidity, Truffle, Ganache, MVC architecture, and other technical domains. These were initially unfamiliar and had to be learned from scratch. This clearly illustrates a high level of enthusiasm for acquiring new knowledge and adapting to new technologies.

While there was a high level of willingness to learn, the implementation of similar functions such as search order and generating PDF invoices in various ways led to the use of different techniques, resulting in some inconsistency in the code. This inconsistency highlights a personal weakness in maintaining uniformity across the project and represents another area for improvement.

8.4.2 Effective Time Management

Secondly, effective time management played a crucial role in the project's success. Given the extensive amount of new material to be learned, a personal timetable was meticulously planned to guide the learning process. Tasks were consistently completed on time, which allowed for ample opportunities to review and refine any imperfect content or code. This proactive approach ensured that the project stayed on track and maintained a high standard of quality throughout its development.

However, good practices in time management for the project did not necessarily translate to effective time management in other areas of daily life. Other aspects of personal management were impacted, illustrating the project's constraints from a different perspective. Balancing project demands with other responsibilities proved challenging, highlighting the need for improved overall time management strategies.

8.4.2 Effective Initial Planning

The initial phase of project planning was effective, as the project scope remained consistent throughout its development. The main direction and functions have stayed aligned with the original plan. This suggests a clear understanding of personal learning abilities and an accurate estimation of the resources, energy, and time required for the project. Consequently, the planning phase set a solid foundation for successful project execution, demonstrating balanced expectations and realistic goal-setting.

Every coin has two sides. The rigidity in sticking to the original plan may have limited opportunities for innovation or adaptation to new insights and technologies that emerged during the project. For instance, the project functions were not particularly significant except for the integration of blockchain. If this personal weakness had been addressed earlier, the project would have had more

opportunities to apply blockchain technology to additional aspects and features, such as using a publicly available SCMS project while only focusing on applying blockchain technology.

In conclusion, this project contributed significantly to the improvement of technical skills as numerous techniques, frameworks, and blockchain-related knowledge were acquired. It also highlighted personal weaknesses and created an awareness for future improvement. Additionally, it enhanced personal confidence due to the ability to overcome various challenges encountered during the project.

8.5 Future Enhancement

There are several areas that highlight the potential for enhancement in WoodChain. By addressing these areas, the system can be made more robust, efficient, and user-friendly.

8.5.1 Consistency

The implemented code should be polished and modified to maintain consistency across the project. This will contribute to the effectiveness of maintenance, updates, and integration of the system in the future. This perspective also concerns naming conventions, code formatting, and file organization. By addressing these areas, the project will be more manageable and scalable, ensuring long-term success and ease of collaboration.

8.5.2 System Design

By delving into the inner workings of WoodChain, it becomes apparent that client-side JavaScript is required to perform certain functions due to the inexperienced system design. For instance, a URL temporary data structure was used for order cart and session management instead of a more robust database solution. While this approach might have seemed easier to implement initially, it unexpectedly posed difficulties during development. The reliance on complex client-side scripts not only increased implementation challenges but also affected code readability and maintainability. Transitioning to a database-driven approach for order cart and session management would improve the system's robustness and simplify the client-side logic.

8.5.3 Testing

The system was tested manually without an automated testing framework for unit tests and integration tests. Implementing an automated testing framework can facilitate early bug detection and ensure that new changes do not break existing functionality. This would enhance the reliability and stability of the system, making it more robust and easier to maintain. For instance, the tool like Istanbul can be used to measuring the test coverage and so on.

8.5.4 User Interface and User Experience

WoodChain also emphasizes the mobility of the application. However, the system is currently not optimized for smaller screen sizes. Improving the layout and content for mobile devices is crucial not

only for visual appeal but also for enhancing the user experience. Additionally, the system and interface design could benefit from incorporating more user feedback to help users become familiar with the system more quickly and effectively.

For instance, the interface for managing orders could prompt the user that the supplier can click to display the invoice and so on. This aspect has been overlooked but could be significantly improved in the future to make the system more intuitive and user-friendly.

By addressing these issues, WoodChain can better meet user needs and provide a seamless experience across different devices and screen sizes.

8.5.5 Security

Although the system applied some security measures such as hash encryption and blockchain technology, additional enhancements are needed to ensure comprehensive security. Improving session management is crucial and can be achieved by utilizing a database to store session information securely. This would help prevent session hijacking and provide better control over user sessions. Data protection should be improved by using HTTPS to encrypt data transmitted between the client and server as well. This helps to protect against eavesdropping and man-in-the-middle attacks.

8.5.6 Functionality

Currently, WoodChain is not specific enough for exclusive use by the customization furniture industry. Although the general applicability of the system is advantageous for broader market potential and revenue generation, it also means that the system does not fully address the unique challenges faced by the customization furniture industry.

To better meet the needs of this industry, the system can be tailored by implementing and improving specific functionalities. For instance, the customization furniture industry is often project-based, requiring multiple orders from different suppliers or repeated orders from the same supplier for a single project. Implementing features that help organize files and orders for specific projects would deeply embed the system into their daily operations, making it more useful for overall management rather than just order purchasing.

For a more intuitive example, the regular user can create a new project in WoodChain, under which all related orders are grouped, indicating that these orders are related to this project. It provides considerable benefits in order tracking, supplier coordination, documentation management, order history reviewing, and financial tracking.

8.5.7 Blockchain Utilization

WoodChain is currently using Ganache, a simulated local Ethereum environment, for blockchain implementation. However, transitioning to a real-world Ethereum environment can provide additional functionalities and benefits.

The blockchain can be further utilized to track the origin and journey of raw materials and finished products throughout the supply chain. This enhances transparency and allows for the verification of the authenticity and quality of materials used. Additionally, blockchain can provide real-time updates to inventory levels, ensuring accurate stock management and reducing the risk of overstocking or stockouts.

Due to the immutable nature of blockchain records, customers can verify the authenticity of products by accessing these records. This feature enhances trust in transactions and assures customers of the product's provenance and integrity. By leveraging these additional blockchain capabilities, WoodChain can offer a more comprehensive and trustworthy supply chain management solution.

8.5.8 Application of Machine Learning

One of the problems listed in Chapter 1 is "Fluctuations in market demand for a specific raw material could lead to insufficient inventory management and product demand estimation.". The application of Machine Learning can address this issue by leveraging predictive analytics to forecast demand for raw materials and finished products. By analyzing historical sales data, market trends, and seasonal patterns, ML algorithms can provide accurate demand predictions. This enables better inventory management, ensuring that sufficient stock levels are maintained to meet demand while minimizing the risk of overstocking. Additionally, ML can help optimize reorder points and stock levels, leading to more efficient and responsive supply chain operations.

Appendix A - References

- Abuzaid, A. N., Alateeq, M. M., Baqleh, L. A., Madadha, S. M., & Haraisa, Y. A. (2023). The moderating effect of strategic momentum on the relationship between big data analytics capabilities and lean supply chain practices. *Uncertain Supply Chain Management*, 11(3), 1–2. <https://doi.org/10.5267/j.uscm.2023.4.013>
- ACTouch. (2023, August 3). *Make-to-Order Vs Make-to-Stock: Great Secrets revealed on MTO and MTS Process*. ACTouch ERP Software. Retrieved November 12, 2023, from <https://www.actouch.com/knowledgebase/make-to-order-vs-make-to-stock/>
- Adobe Communications Team. (2022, March 18). *Waterfall Methodology: Project Management / Adobe Workfront*. Retrieved November 15, 2023, from <https://business.adobe.com/blog/basics/waterfall>
- Akhtar, H. (2023, July 17). *What is White Box Testing? (Example, Types, & Techniques) / BrowserStack*. BrowserStack. Retrieved June 22, 2024, from <https://www.browserstack.com/guide/white-box-testing>
- Alhasawi, E., Hajli, N., & Dennehy, D. (2023). A Review of Artificial intelligence (AI) and Machine learning (ML) for Supply Chain resilience: Preliminary findings. *IEEE Conference Publication / IEEE Xplore*. <https://doi.org/10.1109/ISTAS57930.2023.10306041>
- Altvater, A. (2023, May 2). *What is SDLC? Understand the software development life cycle*. Stackify. Retrieved November 17, 2023, from <https://stackify.com/what-is-sdlc/>
- Amazon. (n.d.). *What is Java? - Java Programming Language Explained - AWS*. Amazon Web Services, Inc. Retrieved November 13, 2023, from <https://aws.amazon.com/what-is/java/>

- Ashtari, H. (2022, September 29). *Black Box Testing vs. White Box Testing - Spiceworks Inc.* Spiceworks Inc. Retrieved June 22, 2024, from <https://www.spiceworks.com/tech/devops/articles/black-box-vs-white-box-testing/>
- Bak, G., & Reicher, R. Z. (2023). Challenges of the SMEs in the 21st century. *Zeszyty Naukowe, 2023*(166), 31–47. <https://doi.org/10.29119/1641-3466.2022.166.3>
- Bayraktar, M., & Algan, N. (2019). The Importance Of SMEs On World Economies. *Uluslararası Avrasya Ekonomileri Konferansı.* <https://doi.org/10.36880/c11.02265>
- Ben-Daya, M., Hassini, E., & Bahroun, Z. (2017). Internet of things and supply chain management: a literature review. *International Journal of Production Research, 57*(15–16), 4719–4742. <https://doi.org/10.1080/00207543.2017.1402140>
- Bhandari, P. (2023, June 22). *What is quantitative research? / Definition, uses & methods.* Scribbr. <https://www.scribbr.com/methodology/quantitative-research/>
- Bhat, A. (2023a, August 10). *Experimental Research: Types of Designs.* QuestionPro. <https://www.questionpro.com/blog/experimental-research/#:~:text=Experimental%20research%20is%20a%20powerful,the%20outcome%20of%20a%20study.>
- Bhat, A. (2023b, September 29). *Questionnaires: The ultimate guide, advantages & examples.* QuestionPro. <https://www.questionpro.com/blog/what-is-a-questionnaire/>
- Blockchain Council. (2024, June 17). *Ganache Blockchain - Blockchain Council.* Retrieved July 12, 2024, from <https://www.blockchain-council.org/blockchain/ganache-blockchain-all-you-need-to-know/>
- Blockchain for supply chain - IBM Blockchain.* (n.d.). <https://www.ibm.com/blockchain-supply-chain>

- Boehm, K. (2023, July 14). *Precoro review 2023: Price, pros & cons*. Software Connect. Retrieved November 15, 2023, from <https://softwareconnect.com/procurement/precoro/>
- Cascading style sheets*. (n.d.). <https://www.w3.org/Style/CSS/Overview.en.html>
- Červený, L., Sloup, R., Červená, T., Riedl, M., & Palátová, P. (2022). Industry 4.0 as an Opportunity and Challenge for the Furniture Industry—A Case Study. *Sustainability*, 14(20), 13325. <https://doi.org/10.3390/su142013325>
- Choudhary, Y. (2023, November 15). *The SDLC models & methodologies: Agile, Scrum, Waterfall*. Finoit Technologies. Retrieved November 17, 2023, from <https://www.finoit.com/blog/sdlc-models-methodologies/#:~:text=Agile%20SDLC%20methodology%20follows%20the,every%20step%20collaboration%20is%20expected>.
- Chris, K. (2021a, August 30). *What is PHP? The PHP Programming Language Meaning Explained*. freeCodeCamp.org. Retrieved November 13, 2023, from <https://www.freecodecamp.org/news/what-is-php-the-php-programming-language-meaning-explained/>
- Chris, K. (2021b, October 7). *What is Go? Golang Programming Language Meaning Explained*. freeCodeCamp.org. <https://www.freecodecamp.org/news/what-is-go-programming-language/>
- Collins, T. (2023, July 26). *What is Grey Box Testing? (Techniques & Example) / BrowserStack*. BrowserStack. Retrieved June 22, 2024, from <https://www.browserstack.com/guide/grey-box-testing>
- Coursera. (2023, June 15). *What Is Python Used For? A Beginner's Guide*. Coursera. Retrieved November 13, 2023, from <https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python>

- De Vass, T., Shee, H., & Miah, S. J. (2020). IoT in supply chain management: a narrative on retail sector sustainability. *International Journal of Logistics: Research and Applications*, 24(6), 605–624. <https://doi.org/10.1080/13675567.2020.1787970>
- Due. (2023, June 19). *Make to Order (MTO) - due*. Retrieved November 12, 2023, from <https://due.com/terms/make-to-order-mto/>
- Duke University Libraries. (n.d.). *LibGuides: Qualitative Research: observation*. Retrieved November 15, 2023, from [https://guides.library.duke.edu/c.php?g=289813&p=1934020#:~:text=What%20is%20an%20observation%3F,they%20are%20being%20watched\).](https://guides.library.duke.edu/c.php?g=289813&p=1934020#:~:text=What%20is%20an%20observation%3F,they%20are%20being%20watched).)
- EduBridge. (2023, October 7). Types of testing: Exploring the types of software testing | Edubridge. *Edubridgeindia*. Retrieved June 22, 2024, from <https://www.edubridgeindia.com/blog/must-know-types-of-software-testing/>
- Ene, C. (2020). Smart contracts - the new form of the legal agreements. *Proceedings of the . . . International Conference on Business Excellence*, 14(1), 1206–1210. <https://doi.org/10.2478/picbe-2020-0113>
- Evjemo, L. D., Gjerstad, T. B., Grøtli, E. I., & Sziebig, G. (2020). Trends in smart manufacturing: Role of humans and industrial robots in smart factories. *Current Robotics Reports*, 1(2), 35–41. <https://doi.org/10.1007/s43154-020-00006-5>
- FarEye. (2023, January 4). *The impact poor supply chain visibility on your bottom line*. Retrieved October 9, 2023, from <https://fareye.com/resources/blogs/the-cons-of-poor-supply-chain-visibility#:~:text=Delays%20in%20delivery%2C%20rising%20logistics,a%20supply%20chain%20domino%20effect>
- GeeksforGeeks. (2023, May 8). *HTML Introduction*. Retrieved November 13, 2023, from <https://www.geeksforgeeks.org/html-introduction/>

George, T. (2023a, June 22). *Mixed Methods Research / Definition, Guide & Examples*.

Scribbr. <https://www.scribbr.com/methodology/mixed-methods-research/>

George, T. (2023b, June 22). *Types of interviews in research / Guide & Examples*. Scribbr.

Retrieved November 15, 2023, from

<https://www.scribbr.com/methodology/interviews-research/#:~:text=An%20interview%20is%20a%20qualitative,the%20interviewer%20asking%20the%20questions>

George, T. (2023c, June 22). *What is a focus group? / Step-by-Step Guide & examples*.

Scribbr. <https://www.scribbr.com/methodology/focus-group/#:~:text=a%20focus%20group%3F-,A%20focus%20group%20is%20a%20research%20method%20that%20brings%20together,of%204%20types%20of%20interviews>

Hamilton, T. (2023, October 7). *V-Model in software testing*. Guru99. Retrieved November 15, 2023, from <https://www.guru99.com/v-model-software-testing.html>

Howell, J. (2023, June 26). *What is Hardhat – A Comprehensive Guide*. 101 Blockchains.

Retrieved November 14, 2023, from <https://101blockchains.com/hardhat-tutorial/>

IBM. (2022, April). *IBM Blockchain Platform Build. Operate. Govern. Grow. Technical Overview*. Retrieved March 13, 2024, from

<https://www.ibm.com/downloads/cas/Q9DGLV7>

IBM. (2023). *Blockchain for supply chain - IBM Blockchain*. Retrieved March 13, 2024, from <https://www.ibm.com/blockchain-supply-chain>

Ilanković, N., Živanić, D., Zelić, A., & Szabó, L. (2019). Basic principles of Industry 4.0 as the foundation for smart factories and digital supply networks. *ResearchGate*.

https://www.researchgate.net/publication/337032679_Basic_principles_of_Industry_4.0_as_the.foundation_for_smart_factories_and_digital_supply_networks

- Infinity, B. & Board Infinity. (2023, July 12). *Prototype Model in Software Engineering / Board Infinity*. Board Infinity. <https://www.boardinfinity.com/blog/a-quick-guide-to-prototype-model-in-software-engineering/>
- Inuwa, M. (2022, December 15). *A Guide to Vyper and its Environments*. Analytics Vidhya. Retrieved November 14, 2023, from <https://www.analyticsvidhya.com/blog/2022/12/a-guide-to-vyper-and-its-environments/>
- Irfan, M., Wang, M., & Akhtar, N. (2019). Enabling supply chain agility through process integration and supply flexibility. *Asia Pacific Journal of Marketing and Logistics*, 32(2), 519–547. <https://doi.org/10.1108/apjml-03-2019-0122>
- Jalli, A. (2022, December 21). *What is Laravel?* Built In. Retrieved November 14, 2023, from <https://builtin.com/software-engineering-perspectives/laravel>
- JavaTPoint. (2023). *Agile Model (Software Engineering) - javatpoint*. [www.javatpoint.com](http://www.javatpoint.com/software-engineering-agile-model). Retrieved November 15, 2023, from <https://www.javatpoint.com/software-engineering-agile-model>
- Jordana. (2023, August 1). *What is JavaScript? A basic introduction to JS for beginners*. Hostinger Tutorials. Retrieved November 13, 2023, from <https://www.hostinger.my/tutorials/what-is-javascript>
- Kanike, U. K. (2023). Factors disrupting supply chain management in manufacturing industries. *journals.open.tudelft.nl*. <https://doi.org/10.18757/jscms.2023.6986>
- Kasauli, R., Knauss, E., Horkoff, J., Liebel, G., & De Oliveira Neto, F. G. (2021). Requirements engineering challenges and practices in large-scale agile system development. *Journal of Systems and Software/≈ the αJournal of Systems and Software*, 172, 110851. <https://doi.org/10.1016/j.jss.2020.110851>

- Katsaliaki, K., Galetsi, P., & Kumar, S. (2021). Supply chain disruptions and resilience: a major review and future research agenda. *Annals of Operations Research*, 319(1), 965–1002. <https://doi.org/10.1007/s10479-020-03912-1>
- Khare, M. (2023, November 7). *What is Node.js and why you should use it*. Kinsta®. Retrieved November 14, 2023, from <https://kinsta.com/knowledgebase/what-is-nodejs/>
- Komalavalli, C., Saxena, D., & Laroiya, C. (2020). Overview of blockchain technology concepts. In *Elsevier eBooks* (pp. 349–371). <https://doi.org/10.1016/b978-0-12-819816-2.00014-9>
- Kuo, T., Rojas, H. Z., & Ohno-Machado, L. (2019). Comparison of blockchain platforms: a systematic review and healthcare examples. *Journal of the American Medical Informatics Association*, 26(5), 462–478. <https://doi.org/10.1093/jamia/ocy185>
- Kushwaha, S. S., Joshi, S., Singh, D., Kaur, M., & Lee, H.-N. (2022). Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract. *IEEE Journals & Magazine / IEEE Xplore*, 6605–6621. <https://doi.org/10.1109/ACCESS.2021.3140091>
- Lawton, G. (2023, August 31). *Top 9 blockchain platforms to consider in 2023*. TechTarget. Retrieved November 13, 2023, from <https://www.techtarget.com/searchcio/feature/Top-9-blockchain-platforms-to-consider>
- Martin, M. (2023, September 19). *Spiral model: When to use? Advantages and disadvantages*. Guru99. Retrieved November 15, 2023, from <https://www.guru99.com/what-is-spiral-model-when-to-use-advantages-disadvantages.html>
- Muheesi, A. (2022). SUPPLY CHAIN MANAGEMENT. *ResearchGate*. <https://doi.org/10.13140/RG.2.2.16353.38241>

Mukta, S. N. (2023). Blockchain Technology: An Overview. *ResearchGate*.

[https://www.researchgate.net/publication/368879391_Blockchain_Technology_An_O
verview#:~:text=Blockchain%20is%20a%20digital%2C%20dec,the%20participants
%20in%20the%20network.](https://www.researchgate.net/publication/368879391_Blockchain_Technology_An_Overview#:~:text=Blockchain%20is%20a%20digital%2C%20dec,the%20participants%20in%20the%20network)

Nanayakkara, S., Rodrigo, M., Perera, S., Weerasuriya, G. T., & Hijazi, A. A. (2021). A methodology for selection of a Blockchain platform to develop an enterprise system.

Journal of Industrial Information Integration, 23, 100215.

<https://doi.org/10.1016/j.jii.2021.100215>

Olusola, S. (2024, June 4). *How to use EJS to template your Node.js application - LogRocket Blog*. LogRocket Blog. Retrieved July 12, 2024, from

<https://blog.logrocket.com/how-to-use-ejs-template-node-js-application/>

Organisation for Economic Co-operation and Development. (2022). *Financing SMEs and Entrepreneurs 2022*. OECD. <https://www.oecd-ilibrary.org/sites/3bc2915c-en/index.html?itemId=/content/component/3bc2915c-en>

Polge, J., Robert, J., & Traon, Y. L. (2021). Permissioned blockchain frameworks in the industry: A comparison. *ICT Express*, 7(2), 229–233.

<https://doi.org/10.1016/j.icte.2020.09.002>

Prata, D. N., Araujo, H. X., & Santos, C. (2021). A Literature Review about Smart Contracts Technology. *International Journal of Advanced Engineering Research and Science*, 8(2), 001–004. <https://doi.org/10.22161/ijaers.82.1>

Samani, N. (2023, April 12). *Furniture manufacturing: Critical issues and challenges*.

Deskera Blog. Retrieved September 30, 2023, from

<https://www.deskera.com/blog/furniture-manufacturing-critical-issues-and-challenges/>

- Sarker, I. H. (2021). Machine learning: algorithms, Real-World applications and research directions. *SN Computer Science*, 2(3). <https://doi.org/10.1007/s42979-021-00592-x>
- Schmitt, J. (2021, December 3). *Unit testing vs integration testing*. CircleCI. Retrieved June 22, 2024, from <https://circleci.com/blog/unit-testing-vs-integration-testing/>
- Sharma, A. (2024, July 11). *Express JS Tutorial*. Simplilearn.com. Retrieved July 12, 2024, from <https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js>
- Simplilearn. (2023, May 26). *What is Solidity Programming: Data Types, Smart Contracts, and EVM?* Simplilearn.com. Retrieved November 14, 2023, from <https://www.simplilearn.com/tutorials/blockchain-tutorial/what-is-solidity-programming>
- Smartbear. (n.d.). *What is unit testing?* smartbear.com. Retrieved June 22, 2024, from <https://smartbear.com/learn/automated-testing/what-is-unit-testing/>
- Song, J. M., Sung, J., & Park, T. (2019). Applications of blockchain to improve supply chain traceability. *Procedia Computer Science*, 162, 119–122. <https://doi.org/10.1016/j.procs.2019.11.266>
- Taherdoost, H. (2023). Smart contracts in Blockchain Technology: A critical review. *Information*, 14(2), 117. <https://doi.org/10.3390/info14020117>
- Team, S. C. (2023, October 30). *What is the Truffle Suite? – Explained*. Shardeum Blogs | EVM-based Sharded L1 Blockchain. Retrieved November 14, 2023, from <https://shardeum.org/blog/truffle/#:~:text=1.-,What%20is%20Truffle%20used%20for%20in%20Blockchain%3F,testing%2C%20deployment%2C%20and%20management>
- .
- Tenny, S. (2022, September 18). *Qualitative study*. StatPearls - NCBI Bookshelf. Retrieved November 15, 2023, from <https://www.ncbi.nlm.nih.gov/books/NBK470395/>

TrustRadius. (2023a, February 28). *Magaya / Technology Without Compromises / leading platform in logistics and supply chain automation* [Video]. TrustRadius. Retrieved November 15, 2023, from <https://www.trustradius.com/products/magaya-supply-chain/reviews?qs=pros-and-cons#product-details>

TrustRadius. (2023b, May 7). *Pros and Cons of FreightPOP 2023*. Retrieved November 15, 2023, from <https://www.trustradius.com/products/freightpop/reviews?qs=pros-and-cons#reviews>

TRY QA. (n.d.). *Diagram of V-model*.

<https://www.google.com/url?sa=i&url=https%3A%2F%2Ftryqa.com%2Fwhat-is-v-model-advantages-disadvantages-and-when-to-use-it%2F&psig=AOvVaw2k6-rWQqQCFS6SQwug9wsr&ust=1700116059866000&source=images&cd=vfe&opi=89978449&ved=0CBIQjRxqFwoTCODQ6IKwxYIDFQAAAAAdAAAAABAD>

Upadhyay, S., Garg, S., & Sharma, R. (2023). Analyzing the factors for implementing Make-to-Order manufacturing system. *Sustainability*, 15(13), 10312.

<https://doi.org/10.3390/su151310312>

Wilson, L. (2019). Quantitative research. In *Springer eBooks* (pp. 27–49).

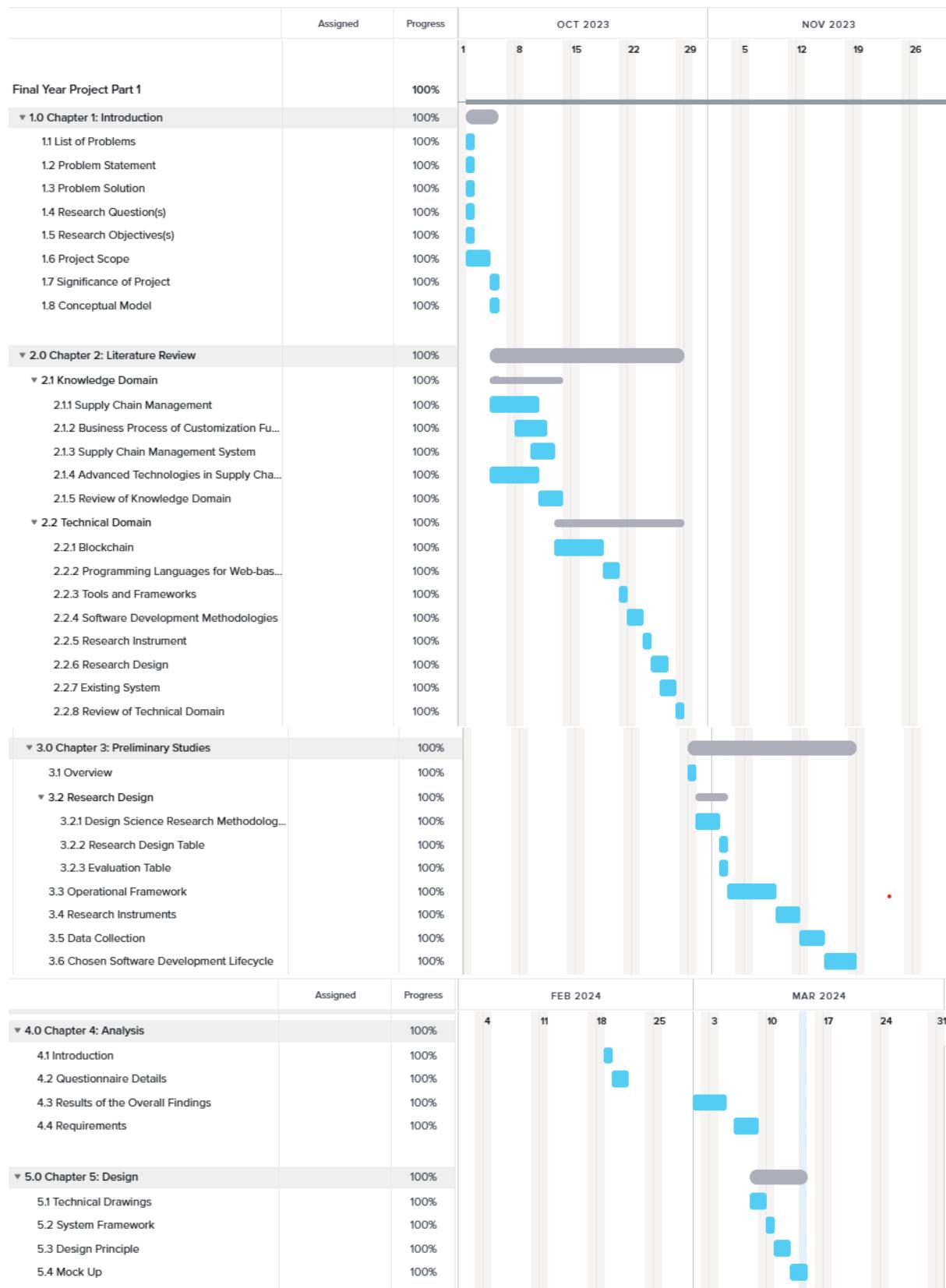
https://doi.org/10.1007/978-981-10-5251-4_54

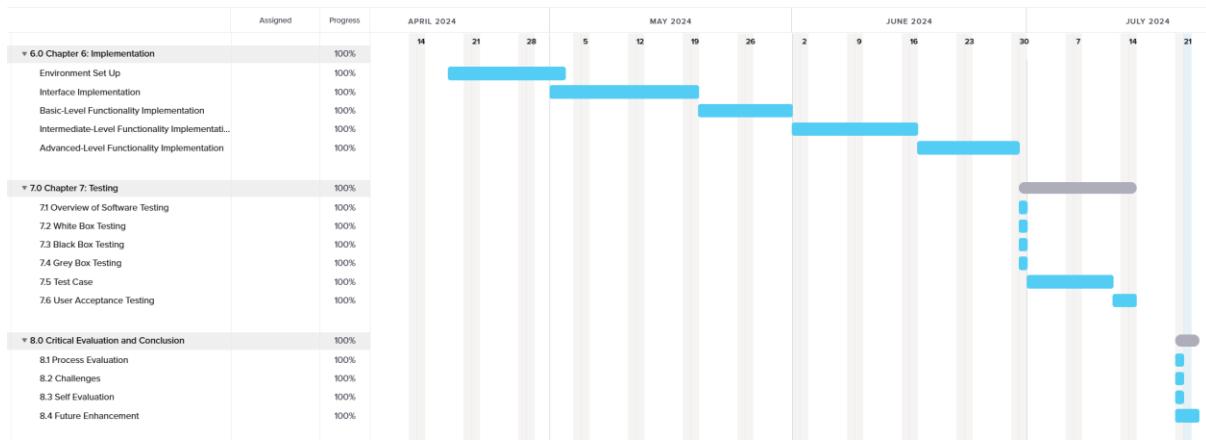
Yasar, K., & Black, R. (2023, March 14). *system testing*. Software Quality. Retrieved June 22, 2024, from <https://www.techtarget.com/searchsoftwarequality/definition/system-testing>

Zola, A. (2022, August 3). *Bootstrap*. WhatIs.com. Retrieved November 14, 2023, from <https://www.techtarget.com/whatis/definition/bootstrap>

Zubair, M. (2023). *Educative Answers - trusted answers to developer questions*. Educative. Retrieved November 14, 2023, from <https://www.educative.io/answers/what-is-web3js>

Appendix B - Gantt Chart





Appendix C - Turnitin Report

turnitin 

Digital Receipt

This receipt acknowledges that **Turnitin** received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author: **Han Cong Goo**
Assignment title: **Fyp2 Documentation**
Submission title: **Goo Han Cong 0133677**
File name: **Goo_Han_Cong_0133677.pdf**
File size: **10.35M**
Page count: **337**
Word count: **50,025**
Character count: **329,050**
Submission date: **26-Jul-2024 04:51PM (UTC+0800)**
Submission ID: **2422675770**

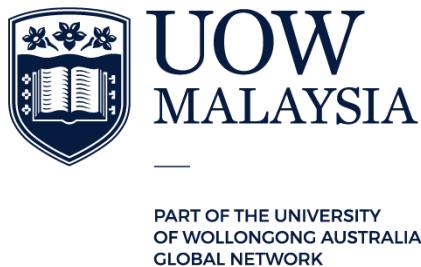
A screenshot of the Turnitin Assignment Coversheet. It includes fields for student name (Han Cong Goo), assignment title (Fyp2 Documentation), and submission date (26-Jul-2024). There are also sections for lecturer name (Mr. Ihsan Firdaus) and subject code (BTEC1001). A QR code is present at the top right.

Copyright 2024 Turnitin. All rights reserved.

> [Fyp2 Documentation](#) 

Paper Title	Uploaded	Grade	Similarity
Goo Han Cong 0133677	26 Jul 2024 16:51	--	 7%

Appendix D – Log Sheet



Department of Computing

Notes on use of the project logbook

1. The purpose of the Project Logbook to document these meetings and therefore build up a record of the student's progress throughout the project.
2. The student should prepare for the meetings by deciding which questions he or she needs to ask the lecturer and what progress has been made since the last meeting (if applicable) and noting these in the relevant sections of the sheet, effectively forming an agenda for the meeting.
3. Log sheets are compulsory assessment criteria for **Final Year Project**. Students who fail to meet the requirements of log sheets will not be allowed to submit **Final Year Project Report**.

Project Title	:	Development of Supply Chain Management System Using Blockchain technology for Customization Furniture Industry
Name	:	Goo Han Cong
ID	:	0133677
Email	:	0133677@student.uow.edu.my
Mobile Number	:	011-18668016
Programme/ Specialization	:	Bachelor of Information System (UEIS)
Main Supervisor	:	Siti Fazilah Shamsudin
Email	:	Fazilah.s@uow.edu.my
Co-Supervisor (If applicable)	:	

Week 5/ Date	Task : Introduction <ul style="list-style-type: none"> • Problem Background • Project Aim • Objectives • Scope • Importance of the project
	Student Meeting Minute/A chieveme nts/Activi ties) <ul style="list-style-type: none"> • Supervisor approved the topic and list of problems. • The problem statement and problem solution had been refined.
	Superviso r (Suggesti on & Comment s) <ul style="list-style-type: none"> • Supervisor suggested the conceptual framework need to be presented with the blockchain architecture. • The project title need to be shorten and more precise. • Supervisor suggested the rest of the parts in this project needs to be more relate to blockchain.
	Next Meeting Plan <ul style="list-style-type: none"> • Studying the existing system and review the chapter 2.
Superviso r's Signature /Date	Siti Fazilah 16-2-2024

Week 6 / Date	Task : Literature Review	
	<ul style="list-style-type: none"> • Introduction • Inter-organization Case Study (if any) <ul style="list-style-type: none"> ◦ To identify user requirements • Current system analysis / Technical Domain (e.g.: product/prototype/software/tools) • Characteristics of the system • Compare between existing systems • Ensure strength and weakness between systems • Literature review on technology used • Chapter summary 	
	Student Meeting Minute/Achievements/Activities)	<ul style="list-style-type: none"> • Conducted an in-depth review of the literature. • The knowledge and technical domain were mostly approved by supervisor. • The exiting system review was mostly approved by the supervisor. • The language, framework, and tools for system development had firstly identified.
	Supervisor (Suggestion & Comments)	<ul style="list-style-type: none"> • Supervisor suggested to find one more existing system which using the blockchain technology. • Supervisor suggested to tabulate the existing system review.
	Next Meeting Plan	<ul style="list-style-type: none"> • Review the questionnaire questions.
	Supervisor's Signature/Date	 19-2-2024

Week 7 / Date	Task : System Development Methodology	
	<ul style="list-style-type: none"> • Introduction • Methodology choice and justification. • Phases within the chosen methodology (traditional or modern) <ul style="list-style-type: none"> ◦ Describes activities and process in each phase ◦ Design modelling (e.g. using UML) ◦ Design tools (e.g. Star UML) ◦ Gantt Chart for FYP 1 and FYP 2 • Describe briefly the technology or tools used to develop the system. • System requirement analysis: hardware and software • List and justify • Chapter summary 	
	Student Meeting Minute/Achievements/Activities)	<ul style="list-style-type: none"> • Questionnaire questions were finalized and allowed to distribute
	Supervisor (Suggestion & Comments)	<ul style="list-style-type: none"> • Supervisor suggested the way to rearrange the questionnaire questions. • Supervisor suggested to add the questions that concern with the willingness, acceptance, trust of using proposed system to questionnaire. • Start to develop the UI as it could contributes to the UML design.
	Next Meeting Plan	<ul style="list-style-type: none"> • Review the conclusion of finding • Review the UML design • Review the whole chapter 4 and 5
	Supervisor's Signature/Date	 26-2-2023

Week 9 / Date	Task : Requirement Analysis and Design	
	<ul style="list-style-type: none"> • Introduction • Requirement analysis <ul style="list-style-type: none"> ◦ OOP (use case, sequence, and activity diagrams) or ◦ Traditional (Software Development Life Cycle) • Design <ul style="list-style-type: none"> ◦ OOP class diagram, overall system architecture or Traditional (DFD) • Database design (if any) <ul style="list-style-type: none"> ◦ ERD (for traditional methodology) ◦ Normalized tables (to include primary key, foreign key, etc.) • Interface design <ul style="list-style-type: none"> ◦ Menu and screen design ◦ System navigation and content design ◦ For web page project to include page navigation • Chapter summary 	
	Student Meeting Minute/Achievements/Activities)	<ul style="list-style-type: none"> • Conclusion of finding was approved by supervisor. • UI was approved.
	Supervisor (Suggestion & Comments)	<ul style="list-style-type: none"> • Since the system is web app. Supervisor suggested to do all the other UML diagrams. • Do one more context diagram and class diagram • Supervisor suggested to do the blockchain architecture and process diagrams. • Supervisor suggested to start the front-end development as it needs to be demonstrated in presentation. • The front-end could do the credential parts first.
	Next Meeting Plan	
	Supervisor's Signature/Date	 11-3-2024

Week 5 / Date	Task : Results, Testing and Discussion <ul style="list-style-type: none"> • Introduction • Coding of system's main functions • Essential interfaces that show system's results and achievements • Testing <ul style="list-style-type: none"> ○ Black box testing <ul style="list-style-type: none"> ▪ System flow, input/output, error messages ○ White box testing ○ User testing • Chapter summary 	
	Student Meeting Minute/Achievements/Activities)	<ul style="list-style-type: none"> • Done the interface implementation • Done the basic functional requirement and doing intermediate functional requirement • Finalized all the elements in system analysis and design
	Supervisor (Suggestion & Comments)	<ul style="list-style-type: none"> • Supervisor suggested to finish all the functional requirement in two weeks. • Supervisor reviewed and accepted the test cases. • Supervisor asked to get the Chapter 8 done before next meeting.
	Next Meeting Plan	<ul style="list-style-type: none"> • Review whole documentation and system.
	Supervisor's Signature/Date	 18-06-2024

Week 10 / Date	Task : Conclusion	
	<ul style="list-style-type: none"> • Introduction <ul style="list-style-type: none"> ◦ Restate the project significance and objectives. • Achievements <ul style="list-style-type: none"> ◦ Briefly explain findings based on literature review ◦ Briefly explain any objectives that has been concluded or partially concluded • Suggested plan for project implementation/execution (FYP 2) 	
	Student Meeting Minute/Achievements/Activities)	<ul style="list-style-type: none"> • Done implementation of all functional requirement • Done the documentation
	Supervisor (Suggestion & Comments)	<ul style="list-style-type: none"> • Supervisor reviewed both system execution and documentation
	Next Meeting Plan	-
	Supervisor's Signature/Date	 21-07-2024