Happy Meal Take-out System Project

Structure of the Final Systems Document

(CEN 5011 Advanced Software Engineering)

Team:	2
Member:	Alex Marrero Pazos
	Jineth Quintero
	Raul Carmine Camarca
	Weitong Liu
	Wenbo Wang
Professor:	Peter Clarke
Date:	April 29, 2016

Abstract

The purpose of this document is to present the third deliverable for the course CEN5011. The document can be divided into four major areas. The first one, chapter one through four, introduces our project, the Happy Meal Take-out System (HMTS), presenting project plan, system functional and nonfunctional requirements, and use case diagrams. The second one, chapters five and six, presents the overall system design through an overview of the software architecture by means of package diagrams, subsystem decomposition, hardware and software mapping, persistent data management strategies and security management protocols. The fourth one, chapter seven, describes the testing process for the units, subsystems, and system, and presents the evaluation of all the tests.

The document also includes a glossary of relevant domain-specific terms, references, and seven appendices intended to provide an overview of the software development process. These include the project schedule, all use cases, user interface designs, detailed class diagrams, class interfaces for the implemented subsystems, documented code for the test driver, and a diary of meetings and tasks.

We believe that our project offers to restaurateurs a cutting-edge advantage over existing systems by meeting the need for intuitive, time-efficient service through a full-featured native mobile customer application and a web client that will enable staff to focus on the quality of products and delivery rather than on the logistics of how to provide their business to their customers. It is our hope that the current document will fully support this belief, not only by underscoring the advantages and benefits offered by HMTS, but also by showcasing its resilient, versatile, and highly adaptable software-hardware infrastructure, having overcome comprehensive and detailed testing through specialized automated testing and code coverage tools.

Table of Contents

1. Introduction

This chapter introduces the reader to the purpose, scope, and development methodology of the Happy Meal Takeout System (HMTS). This chapter is intended as an introduction to provide the reader with the basic system knowledge to thoroughly understand and analyze the proceeding document. This chapter is essential to the understanding of the system; it should be read before the reader attempts to extract any valuable information from subsequent chapters.

1.1. Purpose of system

At its core, the HMTS is intended to be a software solution for restaurants that want to streamline and digitize their ordering and delivery platforms. The system is intended to replace most if not all of the current systems a restaurant is using for these purposes (or encompass their functionalities in the case of a new restaurant). The HMTS is a simple, easy-to-use solution that allows restaurateurs to focus on their products rather than on the logistics of how to provide them to their customers. The overall scope of the system is detailed in Section 1.2. As an outline, the system will encompass customer ordering, menu management, and delivery functionalities. This system, however, is not intended to replace payment or bookkeeping operations.

The intended client of the HMTS is a restaurateur that owns and operates one (or multiple) locations of the same restaurant, offering the same menu. To take advantage of the full functionality spectrum of the HMTS, the restaurant should have food delivery capabilities. In addition to managing restaurant operations via a website, the HTMS will supply the restaurant with a full-featured native mobile application that will allow the restaurant's customers to place orders. The HTMS system will also supply restaurants with a mobile application develop for its delivery staff, whose purpose is to streamline the delivery process, increase order tracking functionalities, and help delivery drivers navigate their routes more easily.

The system is described as offering a "streamlined" and "digital" solution because it provides the basic system functionalities needed to successfully operate a restaurant business in easy-to-use interfaces in a completely digital space (i.e., no paperwork necessary).

1.2. Scope of system

As outlined in section 1.1, the overall system encompasses customer ordering, menu management, and delivery functionalities. The native mobile application features the customer interface, which encompasses the ordering functionalities (e.g., viewing menu, adding product to cart, checking out, entering payment info, etc.). The mobile client also features the deliveryman interface, where those employees designated as delivery staff can see orders assigned to them and mark them as delivered. The web client of the system features the functionalities assigned to the operational staff of the restaurant. These functionalities include managing the menu, managing users (managers only), processing orders, etc.

It is important to point out what is outside of the scope of the HMTS. The system does not include any payment processing information. The system collects credit card information for credit card orders and forwards it to a payment processing service. The HMTS also does not include any functionalities for employee time-keeping; that is, there is no way for employees logging "Time-In" and "Time-out" in reference to hours worked. Similar functionalities such as employee performance reviews and deliveryman performance measures (e.g. average delivery times) are also outside of the scope of the system, along with payroll and ledger functionalities. The only ledger functionalities that will occur within the HMTS are payment confirmation and order total tabulation and storage. The system retains order information in persistent storage, and this includes financial information such as order total and item breakdown. Any other system can use this information for other purposes, such as general ledger compilation or inventory management.

It is also important to note that each deployment of the HMTS will be for one restaurant only. There is no functionality to display multiple restaurants' menus so that customers have choices. These functionalities are similar to those of existing services such as GrubHub. These functionalities are not only outside of our scope but reflect a different system purpose altogether. Comparisons to existing services in our market space are discussed in Chapter 2.

1.3. Development methodology

The development of the HMTS followed the phases delineated by the Unified Software Development Model (USDP) shown in Figure 1.1.

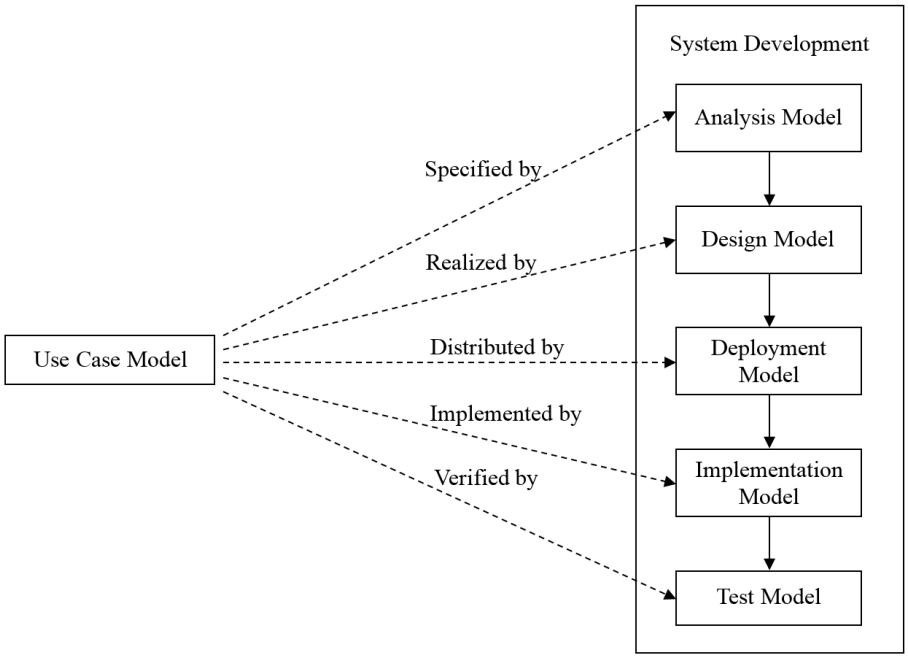


Figure 1.1. Unified Software Development Process.

The Use Case model was first specified by an analysis model (as described in the first deliverable) in which all the requirements were acquire and analyzed to produce the use case diagrams for the system with the help of the sequence diagrams, object diagrams, and the first view of the class diagram..

The Use Case model was then realized by a design model (as described in the second deliverable) through various UML diagrams such as architecture diagrams representing different subsystems, class diagrams detailing subsystem classes, attributes, and operations, minimal class diagrams providing a high-level view of the system, sequence diagrams presenting the steps involved in specific use cases, and state machine diagrams describing system response to external and internal events.

We have now entered in the last phases of the development, in which the Use Case model is distributed by a deployment model and implemented by an implementation model.

In developing the deployment model, we represented the hardware/software mapping of the system. This aided the physical modeling of the system. The deployment diagrams show what hardware components exist, what software components (“artifacts”) run on each node, and how the different components are connected [4].

In developing the implementation model, we translated the solution into source code. This included implementing the attributes and methods of each object and integrating all the objects so that they function as a single system.

1.4. Definitions, acronyms, and abbreviations

This section provides some of the most relevant terms that the reader needs to know in order to more easily grasp the functionality of the HMTS in the following chapters.

- *Actor*: Any individual or external entity interacting with the HMTS.
- *HMTS*: Happy Meal Take-out System.
- *Deliveryman*: Restaurant employee (male or female) that is tasked with delivering products to customers. Will interact with the system mainly through the mobile application.
- *Manager*: Any restaurant employee with managerial privileges; manager can manage employee accounts in the system, alter menu, etc. They will interact with the system exclusively through the web page.
- *Staff*: The staff designation refers to any employee of a restaurant who does not fall into the Deliveryman or Manager categories. Will interact with the system exclusively through the web page.
- *Customer*: An individual who wishes to purchase food from a restaurant where the system is implemented.
- *Product*: Any food, side, or drink item available for sale by the restaurant; separated by Types, but all visualized on the same menu.
- *Roles*: Specific user groupings based on functionality; Manager, Customer, Deliveryman, Staff; used to specify Actors according to their privileges in the system.
- *User*: A generic entity in storage that encompasses all user roles. Corresponds to a customer, manager, deliveryman, or staff member registered in the system.
- *User (Reg)*: Actors that are allowed to create a user account.
- *User (No Reg)*: Actors that are not allowed to create a user account.
- *Artifact*: Specification of a physical piece of information used or produced by a software development process, or by deployment and operation of the system.
- *Node*: Computational resource upon which UML artifacts may be deployed for execution.

1.5. Overview of document

The document highlights the deployment, implementation, and testing stages of the software engineering process for the HMTS.

In Chapter 2 we describe the current system limitations and problems through a brief market analysis.

In Chapter 3 we describe project planning, including project organization, hardware and software requirements, work breakdown, and cost estimate.

In Chapter 4 we describe the requirements of the system, including functional and nonfunctional requirements and multiple-level use case diagrams. We also describe requirements analysis presenting static and dynamic analysis models such as object, class, and sequence diagrams.

In Chapter 5 we describe the overall system design by introducing the software architecture through package diagrams and subsystem decomposition, as well as a deployment diagram describing hardware and software mapping, the persistent data management needs, and the security management protocols.

In Chapter 6 we describe the detailed design of the system including minimal class diagrams for the subsystems, state machine diagrams, sequence diagrams showing object interactions, and detailed class design including OCL for class invariants in the major subsystem controllers.

In Chapter 7 we review the entire testing process, including system, subsystem, and unit test cases. We describe automated testing tools and code coverage tools employed. We also provide the test results and comments intended to address bugs.

In Chapter 8 we provide a Glossary of Terms defining domain-specific terms used throughout the document.

In Chapter 9 we provide the approval page, also known as contract page, signed by all the developers of the HMTS.

In Chapter 10 we provide relevant references quoted through the document.

In Chapter 11 we provide a comprehensive set of seven appendices to be considered supplemental information rather than standalone documentation, including detailed project schedule with Gantt chart, use and misuse cases, user interface design detail class diagrams, code for class interfaces, documented code for test drivers, and a diary of meetings and tasks.

2. Current System

Various systems exist that provide similar functionalities to that brought forth by the HTMS. Such systems include:

- Gloria Foods – offers a service where they will add a widget to your website so that customers can order food. The service will also add a widget to the restaurant's Facebook profile so that customers can access the ordering functionality directly from Facebook. Drawbacks to this system include that it does not have a native mobile application for ordering. Although the system is free, it does have a monthly premium that must be paid to handle billing information. The biggest drawback to this service however is that it requires an already functioning website for the restaurant, which can be a huge overhead cost if the restaurant is just getting started with their digital campaign.

- ChowNow – ChowNow offers a service where they will add an ordering widget to the customer's website. They will also integrate ordering information into the customer's Facebook page. One key advantage of using ChowNow is that it provides market data & trends and helps organize the restaurant's marketing strategy. ChowNow will also develop a custom mobile application for your business, including support. The ChowNow system will cost a restaurant around $2,600 to implement, for 2 years of support and functionality. As a tech startup, ChowNow has recently partnered with fellow tech startup Uber to implement their new UberRush functionality into the ChowNow framework. ChowNow has also implemented Google and Apple pay interfaces to handle payment information. Disadvantages to the ChowNow platform are that it is not very flexible in its implementation; it does not integrate well with existing delivery methods; and it can get cost prohibitive as users add features or restaurant locations. The cost of the system described only covers one location, and adding additional locations essentially counts as adding separate restaurants in the ChowNow pricing model. In order to implement the ChowNow system, the restaurant would already need to have a website in place, which can be cost prohibitive to new businesses in itself. Although partnering with UberRush would allow restaurants using the ChowNow system to deliver product to customers, it does little to help restaurants with delivery staff already on payroll. It also does not help restaurants in non-urban areas where UberRush is not available. Overall,

this system would not be very beneficial to established restaurants with multiple locations and delivery staff, where the Software-as-a-Service business model might not be the most beneficial.

By comparison, the HMTS will be very helpful to established businesses because it allows more flexibility in terms of its implementation. Customers who want to use their own servers would be welcomed to, and will liberate them from high monthly premiums. Although the upfront cost of HMTS would potentially be higher, it might be a sensible business decision to some restaurants looking to upgrade their ordering systems (e.g., if a restaurant takes out a sizable loan for renovations, it can potentially lump system costs with the renovation expenses).

The last point that sets apart the HMTS from other systems is the native application that is developed specifically for delivery staff. This provides the benefit of simplicity for the delivery staff, and facilitates communication between delivery staff and restaurant operational staff. Additional functionalities could be added on request to the delivery application. For example, the application could have a Maps API that would show the deliveryman the best route without him having to resort to a third party GPS device (e.g., Uber uses a simple map API for its drivers so that everything operates within the Uber application.).

The application could also use an algorithm to map the route for the delivery staff so that it is the most economical (e.g., not unlike the way home deliveries like Fedex and UPS map routes beforehand).

3. Project Plan

In this chapter, it indicates a formal, approved document used to guide both project execution and project control. It includes project organization, hardware and software requirements, work breakdown and cost estimate. Project organization describes each team member and his role in the team for each phase. Hardware and software requirements describes the environment of development and runtime. Work breakdown describes the concreted tasks, deliverables and milestones of the project. Cost Estimate is used to compute the necessary effort for the project.

3.1. Project Organization

In this section, we describe roles of each team member in the following three phases. Every team member can have multiple roles in each phase. In Table 3.1 we describe the period of phase 1, team members' names and the project team organization for phase 1.

Table 3.1. Team organization for phase 1.

Phase 1: Jan. 12, 2016 – Feb. 23, 2016		
Wenbo Wang	Leader	Developer
Raul Camarca	Minute Taker	Developer
Jineth Quintero	Timekeeper	UI Designer
Alex Marrero Pazos	Requirement Engineer	Tester
Weitong Liu	Documentation Engineer	Editor

In Table 3.2 we describe the period of phase 2, team members' names and the project team organization for phase 2.

Table 3.2. Team organization for phase 2.

Phase 2: Feb. 24, 2016 – Mar. 29, 2016		
Jineth Quintero	Leader	Editor
Alex Marrero Pazos	Minute Taker	Developer
Weitong Liu	Timekeeper	Developer
Wenbo Wang	System Architect	Developer
Raul Camarca	Developer	Tester

In Table 3.3 we describe the period of phase 3, team members' names and the project team organization for phase 3.

Table 3.3. Team organization for phase 3.

Phase 3: Mar. 30, 2016 – Apr. 26, 2016		
Weitong Liu	Leader	Tester
Jineth Quintero	Minute Taker	Tester
Raul Camarca	Timekeeper	Tester
Alex Marrero Pazos	Developer	Editor
Wenbo Wang	Tester	Developer

3.2. Hardware and Software Requirements

In this section we describe hardware and software environments needed by the system. For hardware environment, we lists hardware component names, models and technical parameters.

1. Hardware requirements:

- CPU: Intel Core i5-4200U Processor 1.6GHz 3MB
- Memory: 4.0GB PC3-12800 DDR3L 1600MHz
- Hard Drive: 128GB SSD
- Graphics: Intel HD 4400
- Network Card: Intel 7260 b/g/n Wireless

For software environment, we lists software platform and tools names, versions and producer names.

2. Software requirements:

- Microsoft Windows 8.1 (OS)
- Microsoft Office Word 2013
- Microsoft Office PowerPoint 2013
- Microsoft Office Project 2013
- Java Platform Version 7 Update 79
- Eclipse Mars.1 Release (4.5.1)
- Papyrus 1.1.3 Released
- Android Studio 1.4.1

3.3. Work Breakdown

In this section we describe concrete tasks for requirements, analysis, design, implement and testing phase. Each task contains six fields, Task Index, Name, Duration, Dependencies, Deliverables and Milestone. In Table 3.4 we describe concrete tasks for requirements and analysis phase. This phase takes forty three days in total. The milestone is "Presentation in Class" task.

Table 3.4. Tasks for requirements and analysis phase.

Task	Name	Duration	Dependencies	Deliverables	Milestone
1	**Req. & Analysis Phase**	**43 days**			
2	Project Selection	5 days			
3	Project Plan	2 days	2	Schedule files	
5	Use Cases Confirmation	1 day	3	Confirmation document	
6	Use Cases	4 days	5	Use cases document & UI	
7	Use Case Diagrams	3 days	6	Use case diagrams	
8	Scenarios	2 days	7	Scenarios document	
10	Object Diagrams	4 days	8	Object diagrams	
11	Class Diagrams	6 days	10	Class diagrams	
12	Dynamic Model	4 days	11	Sequence diagrams	
13	SRD Document	11 days	12	SRD file	
14	Presentation PPT	6 days	13	PPT file	
15	Presentation in Class	1 day	14		√

In Table 3.5 we describe concrete tasks for design phase. This phase takes thirty five days in total. The milestone is "Presentation with Professor" task.

Table 3.5. Tasks for design phase.

Task	Name	Duration	Dependencies	Deliverables	Milestone
16	**Design Phase**	**35 days**			
17	Architectural Patterns	4 days	15	Architecture diagrams	
18	Deployment Diagram	3 days	17	Deployment diagram	
19	ER Diagram	3 days	15	ER diagram	
20	Minimal Class Diagram	5 days	17,19	Minimal class diagrams	
21	State Machines	4 days	17	State Machine diagrams	
22	Sequence Diagrams	4 days	19	Sequence diagrams	
23	Detail Class Diagrams	5 days	20	Detailed class diagrams	
24	Class Interfaces (Code)	5 days	23	Code	
25	OCL	4 days	24	OCL diagram	
26	DD Document	5 days	18,21,22,25	DD file	
27	Presentation with Prof.	7 days	26		√

In Table 3.6 we describe concrete tasks for implementation and testing phase. This phase takes twenty eight days in total. The milestone is “Final Deliverable” task.

Table 3.6. Tasks for implementation and testing phase.

Task	Name	Duration	Dependencies	Deliverables	Milestone
28	**Impl. and Testing Phase**	28 days			
29	Unit Test Cases	2 days	27	Unit test case document	
30	Subsystem Test Cases	2 days	29	Subsystem test case document	
31	System Test Cases	2 days	30	System test case document	
32	Implementation	12 days	27	Code	
33	Unit Test	1 day	29,32	Unit test case report	
34	Subsystem Test	1 day	30,32	Subsystem test case report	
35	System Test	1 day	31,32	System test case report	
36	Test Results	1 day	33,34,35	Test report	
37	Code Coverage	1 day	33,34,35	Document	
38	Final Document	8 days	36,37	Final file	
39	Presentation PPT	3 days	38	PPT file	
40	Presentation in Class	1 day	39		
41	User's Guide	3 days	39,40	User's guide file	
42	Instruction	3 days	32	Instruction file	
43	Impl. Document	3 days	32	Implementation File	
44	Testing Document	3 days	36	Testing file	
45	Final Deliverable	4 days	32,38,41,42,43,44	DVD (Containing all the things)	√

3.4. Cost Estimate

Following functionalities are counted while counting the function points of the system.

- Data Functionality
    - Internal Logical Files (ILF)
    - External Interface Files (EIF)
- Transaction Functionality
    - External Inputs (EI)
    - External Outputs (EO)
    - External Queries (EQ)

According to the requirements of the system and our experience, the function point standard are made and shows as followings. In Table 3.7 we describe the standard for function points of External Inputs. It includes File Type Referenced and Data Elements field.

Table 3.7. External Inputs (EI).

File Type Referenced (FTR)	Data Elements (DET)		
	1-4	5-15	Greater than 15
Less than 2	Low (2)	Low (3)	Average (4)
2	Low (3)	Average (4)	High (5)
Greater than 2	Average (4)	High (5)	High (6)

In Table 3.8 we describe the standard for function points of External Outputs. It includes File Type Referenced and Data Elements field.

Table 3.8. External Outputs (EO).

File Type Referenced (FTR)	Data Elements (DET)		
	1-5	6-19	Greater than 19
Less than 2	Low (2)	Low (2)	Average (4)
2 or 3	Low (4)	Average (5)	High (6)
Greater than 3	High (4)	High (5)	High (6)

In Table 3.9 we describe the standard for function points of External Inquiry. It includes File Type Referenced and Data Elements field.

Table 3.9. External Inquiry (EQ).

File Type Referenced (FTR)	Data Elements (DET)		
	1-5	6-19	Greater than 19
Less than 2	Low (2)	Low (3)	Average (4)
2 or 3	Low (3)	Average (4)	High (6)
Greater than 3	Average (4)	High (5)	High (6)

In Table 3.10 we describe the standard for function points of Internal Logical File. It includes Record Element Types and Data Elements field.

Table 3.10. Internal Logical File (ILF).

Record Element Types (RET)	Data Elements (DET)		
	1-19	20-50	51 or More
1	Low (2)	Low (4)	Average (6)
2 to 5	Low (4)	Average (6)	High (8)
6 or more	Average (5)	High (6)	High (8)

In Table 3.11 we describe the standard for function points of External Interface File. It includes Record Element Types and Data Elements field.

Table 3.11. External Interface File (EIF).

Record Element Types (RET)	Data Elements (DET)		
	1-19	20-50	51 or More
1	Low (2)	Low (4)	Average (6)
2 to 5	Low (4)	Average (6)	High (8)
6 or more	Average (6)	High (8)	High (8)

The values of ILF, EIF, EI, EO and EQ are estimated for each use case according to tables above.

Table 3.12. Estimate for each use case.

NO.	Use Case ID	ILF	EIF	EI	EO	EQ
1	HMTS_UM01_CreateAccount	2	0	3	2	2
2	HMTS_UM02_Login	4	0	2	2	2
3	HMTS_UM04_Logout	2	0	2	2	2
4	HMTS_UM07_RolesControl	2	0	2	2	3
5	HMTS_OP01_VisualizeProducts	2	0	2	5	2
6	HMTS_OP02_AddProduct	2	0	4	2	2
7	HMTS_CU01_VisualizeMenu	2	0	2	5	4
8	HMTS_CU02_AddProduct	4	0	5	2	2
9	HMTS_CU03_ViewCart	4	0	2	5	4
10	HMTS_CU04_RemoveProduct	4	0	2	2	2
11	HMTS_CU05_FillOutOrderInfo	2	0	3	2	2
12	HMTS_CU07_PopAddressAlert	2	0	2	2	2
Total		32	0	31	33	29

The UFP (Unadjusted Function Points) = 32 + 0 + 31 + 33 + 29 = 125 (Sum of all the Complexities of all the EI's, EO's EQ's, ILF's and EIF's of all use cases.). The system is developed for a small business; it doesn’t involve in complex and plentiful data, so the Data Base Size of the system is low and the Product Complexity is low, whereas the Developed for Reusability is high because architecture and design patterns will be used onto the system.

COCOMO II - Constructive Cost Model

Software Size Sizing Method: Function Points

Unadjusted Function Points: 125 Language: Java

Software Scale Drivers

Precedentedness	Nominal	Architecture / Risk Resolution	Nominal	Process Maturity	Nominal
Development Flexibility	Nominal	Team Cohesion	Nominal		

Software Cost Drivers

Product

Required Software Reliability	Nominal
Data Base Size	Low
Product Complexity	Low
Developed for Reusability	High
Documentation Match to Lifecycle Needs	Nominal

Personnel

Analyst Capability	Nominal
Programmer Capability	Nominal
Personnel Continuity	Nominal
Application Experience	Nominal
Platform Experience	Nominal
Language and Toolset Experience	Nominal

Platform

Time Constraint	Nominal
Storage Constraint	Nominal
Platform Volatility	Nominal

Project

Use of Software Tools	Nominal
Multisite Development	Nominal
Required Development Schedule	Nominal

Maintenance Off

Software Labor Rates

Cost per Person-Month (Dollars): 1000

Calculate

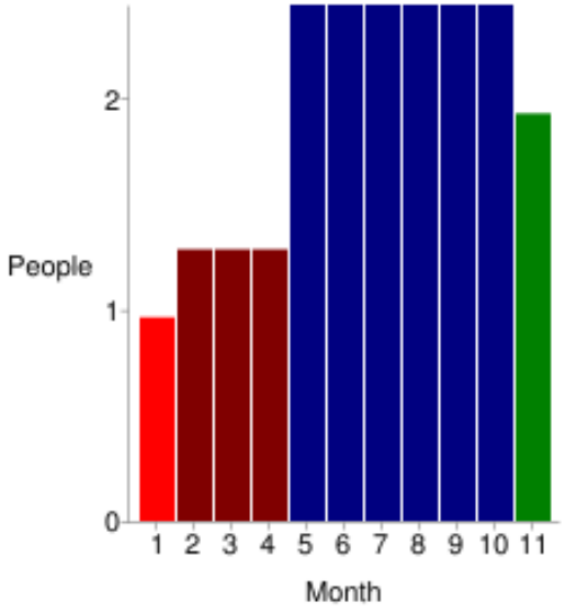
Results

Software Development (Elaboration and Construction)

Effort = 19.7 Person-months
Schedule = 9.8 Months
Cost = $19703

Total Equivalent Size = 6625 SLOC

Staffing Profile



Acquisition Phase Distribution

Phase	Effort (Person-months)	Schedule (Months)	Average Staff	Cost (Dollars)
Inception	1.2	1.2	1.0	$1182
Elaboration	4.7	3.7	1.3	$4729
Construction	15.0	6.1	2.4	$14975
Transition	2.4	1.2	1.9	$2364

Software Effort Distribution for RUP/MBASE (Person-Months)

Phase/Activity	Inception	Elaboration	Construction	Transition
Management	0.2	0.6	1.5	0.3
Environment/CM	0.1	0.4	0.7	0.1
Requirements	0.4	0.9	1.2	0.1
Design	0.2	1.7	2.4	0.1
Implementation	0.1	0.6	5.1	0.4
Assessment	0.1	0.5	3.6	0.6
Deployment	0.0	0.1	0.4	0.7

4. Requirements of System

In this chapter, we first describe the functional requirements, which are the functions the system will support. We then introduce the nonfunctional requirements of our project, including usability, reliability, performance, supportability and implementation. We finally describe the details of the use cases with their corresponded diagrams, which consist of the top level diagram and diagrams in each package.

4.1. Functional and Nonfunctional Requirements

In this section, we talk about the functional and nonfunctional requirements for the whole system. There are twenty five use cases in total.

4.1.1. Functional Requirements

The following requirements contains all of the functionalities of the HMTS:

1. The system shall allow users to create accounts (*see use case HMTS_UM01_CreateAccount in appendix B*).
2. The system shall allow users to login (*see use case HMTS_UM02_Login in appendix B*).
3. The system shall allow users to recover the password in case they forgot it (*see use case HMTS_UM03_ForgetPassword in appendix B*).
4. The system shall allow users to logout the system (*see use case* HMTS_UM04_Logout *in appendix B*).
5. The system shall log out the user from the system due to inactivity (*see use case HMTS_UM05_LoginTimeout in appendix B*).
6. The system shall limit the attempts to login with wrong credentials (*see use case HMTS_UM06_LimitFailedLoginTimes in appendix B*).
7. The system shall allow the user to see only the functionality according to the privileges granted by its role (*see use case HMTS_UM07_RolesControl in appendix B*).
8. The system shall allow manager and staff to visualize a list of products (*see use case HMTS_OP01_VisualizeProducts in appendix B*).
9. The system shall allow manager and staff to add products (dishes, sides or drinks) to the menu (*see use case HMTS_OP02_AddProduct in appendix B*).

10. The system shall allow manager and staff to remove products from the menu (*see use case HMTS_OP03_RemoveProduct in appendix B*).
11. The system shall allow manager and staff visualize orders (*see use case HMTS_OP04_VisualizeOrders in appendix B*).
12. The system shall allow manager and staff to view detailed order information (*see use case HMTS_OP05_VisualizeDetailedOrder in appendix B*).
13. The system shall allow manager and staff to process orders for delivery (*see use case HMTS_OP06_ProcessOrder in appendix B*).
14. The system shall allow customers to visualize the menu (*see use case HMTS_CU01_VisualizeMenu in appendix B*).
15. The system shall allow the customer to add products to a shopping cart (*see use case HMTS_CU02_AddProduct in appendix B*).
16. The system shall allow the customer to view the shopping cart (*see use case HMTS_CU03_ViewCart in appendix B*).
17. The system shall allow the customer to remove products from the shopping cart (*see use case HMTS_CU04_RemoveProduct in appendix B*).
18. The system shall allow the customer to fill out order information and request the placement of an order (*see use case HMTS_CU05_FillOutOrderInfo in appendix B*).
19. The system shall allow the customer to see a confirmation page for the placed order (*see use case HMTS_CU06_VisualizePOC in appendix B*).
20. The system shall allow the customer to see an alert if the system detects an inconsistency between the billing and shipping address for security reasons (*see use case HMTS_CU07_PopAddressAlert in appendix B*).
21. The system shall allow the customer to place an order (*see use case HMTS_CU08_PlaceOrder in appendix B*).
22. The system shall allow the customer to visualize the status of an order (*see use case HMTS_CU09_VisualizeOrderStatus in appendix B*).
23. The system shall allow the delivery man to visualize the orders that are pending (*see use case HMTS_DE01_VisualizeOrders in appendix B*).
24. The system shall allow the delivery man to visualize detailed order information (*see use case HMTS_DE02_VisualizeDetailedOrder in appendix B*).

25. The system shall allow the delivery man to update the status of an order (*see use case HMTS_DE03_ProcessOrder in appendix B*).

The use case diagrams can be found in Section 4.2. The user interfaces of web and mobile application can be found in Appendix C.

4.1.2. Nonfunctional Requirements

This section describes user level requirements not directly related to functionality. The detailed description of the uses cases can be found in Appendix B.

Usability

- No previous training is needed: All use cases.

Reliability

- Mean time to failure – 1 failure for every 24 hours of operation:
  HMTS_UM01_CreateAccount; HMTS_UM02_Login;
  HMTS_UM03_ForgetPassword; HMTS_UM04_Logout;
  HMTS_UM05_LoginTimeOut; HMTS_UM06_LimitFailedLoginTimes;
  HMTS_UM07_RolesControl; HMTS_OP01_VisualizeProducts;
  HMTS_OP02_AddProduct; HMTS_OP03_RemoveProduct;
  HMTS_OP04_VisualizeOrders; HMTS_OP05_VisualizeDetailedOrder;
  HMTS_OP06_ProcessOrder; HMTS_CU01_VisualizeMenu;
  HMTS_CU02_AddProduct; HMTS_CU04_RemoveProduct;
  HMTS_CU03_ViewCart; HMTS_CU07_PopAddressAlert;
  HMTS_CU06_VisualizePOC; HMTS_DE01_VisualizeOrders;
  HMTS_DE02_VisualizeDetailedOrder; HMTS_DE03_ProcessOrder
- Mean Time to Failure – 1 failure for every 72 hours of operation is acceptable: HMTS_CU05_FillOutOrderInfo.
- Mean Time to Failure – 1 failure for every 100 hours of operation is acceptable: HMTS_CU08_PlaceOrder.
- Mean Time to Failure – 1 failure every 12 hours of operation is acceptable: HMTS_CU09_VisualizeOrderStatus.

Performance

- The process should take no more than 2 seconds:
  HMTS_UM01_CreateAccount; HMTS_UM02_Login;
  HMTS_UM03_ForgetPassword; HMTS_UM04_Logout;
  HMTS_UM05_LoginTimeOut; HMTS_UM06_LimitFailedLoginTimes;
  HMTS_UM07_RolesControl; HMTS_OP01_VisualizeProducts;
  HMTS_OP02_AddProduct; HMTS_OP03_RemoveProduct;
  HMTS_OP04_VisualizeOrders; HMTS_OP05_VisualizeDetailedOrder;
  HMTS_CU01_VisualizeMenu; HMTS_CU03_ViewCart;
  HMTS_CU04_RemoveProduct; HMTS_CU05_FillOutOrderInfo;
  HMTS_CU08_PlaceOrder; HMTS_DE01_VisualizeOrders;
  HMTS_DE02_VisualizeDetailedOrder; HMTS_DE03_ProcessOrder.

- The product will be added to a cart in 1 seconds:
  HMTS_CU02_AddProduct; HMTS_CU07_PopAddressAlert;
  HMTS_CU09_VisualizeOrderStatus.

Supportability

- The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed: All use cases.

Implementation

- The implementation of the system is to be done in Java in the Eclipse development environment:
  HMTS_UM01_CreateAccount; HMTS_UM05_LoginTimeOut;
  HMTS_OP01_VisualizeProducts; HMTS_OP02_AddProduct;
  HMTS_OP03_RemoveProduct; HMTS_OP04_VisualizeOrders;
  HMTS_OP05_VisualizeDetailedOrder; HMTS_OP06_ProcessOrder.

- The implementation of the system will be done in Java and Eclipse framework and Android framework:
  HMTS_UM02_Login; HMTS_UM03_ForgetPassword;
  HMTS_UM04_Logout; HMTS_UM06_LimitFailedLoginTimes;

HMTS_UM07_RolesControl; HMTS_CU01_VisualizeMenu;
HMTS_CU02_AddProduct; HMTS_CU03_ViewCart;
HMTS_CU04_RemoveProduct; HMTS_CU05_FillOutOrderInfo;
HMTS_CU06_VisualizePOC; HMTS_CU07_PopAddressAlert;
HMTS_CU08_PlaceOrder; HMTS_CU09_VisualizeOrderStatus;
HMTS_DE01_VisualizeOrders; HMTS_DE02_VisualizeDetailedOrder;
HMTS_DE03_ProcessOrder.

This section describes user level requirements not directly related to functionality. The detailed description of the uses cases can be found in Appendix B.

Usability

- No previous training is needed: All use cases.

Reliability

- Mean time to failure – 1 failure for every 24 hours of operation:
  HMTS_UM01_CreateAccount; HMTS_UM02_Login;
  HMTS_UM03_ForgetPassword; HMTS_UM04_Logout;
  HMTS_UM05_LoginTimeOut; HMTS_UM06_LimitFailedLoginTimes;
  HMTS_UM07_RolesControl; HMTS_OP01_VisualizeProducts;
  HMTS_OP02_AddProduct; HMTS_OP03_RemoveProduct;
  HMTS_OP04_VisualizeOrders; HMTS_OP05_VisualizeDetailedOrder;
  HMTS_OP06_ProcessOrder; HMTS_CU01_VisualizeMenu;
  HMTS_CU02_AddProduct; HMTS_CU04_RemoveProduct;
  HMTS_CU03_ViewCart; HMTS_CU07_PopAddressAlert;
  HMTS_CU06_VisualizePOC; HMTS_DE01_VisualizeOrders;
  HMTS_DE02_VisualizeDetailedOrder; HMTS_DE03_ProcessOrder
- Mean Time to Failure – 1 failure for every 72 hours of operation is acceptable:
  HMTS_CU05_FillOutOrderInfo.
- Mean Time to Failure – 1 failure for every 100 hours of operation is acceptable:
  HMTS_CU08_PlaceOrder.
- Mean Time to Failure – 1 failure every 12 hours of operation is acceptable:

HMTS_CU09_VisualizeOrderStatus.

Performance

- The process should take no more than 2 seconds:
  HMTS_UM01_CreateAccount; HMTS_UM02_Login;
  HMTS_UM03_ForgetPassword; HMTS_UM04_Logout;
  HMTS_UM05_LoginTimeOut; HMTS_UM06_LimitFailedLoginTimes;
  HMTS_UM07_RolesControl; HMTS_OP01_VisualizeProducts;
  HMTS_OP02_AddProduct; HMTS_OP03_RemoveProduct;
  HMTS_OP04_VisualizeOrders; HMTS_OP05_VisualizeDetailedOrder;
  HMTS_CU01_VisualizeMenu; HMTS_CU03_ViewCart;
  HMTS_CU04_RemoveProduct; HMTS_CU05_FillOutOrderInfo;
  HMTS_CU08_PlaceOrder; HMTS_DE01_VisualizeOrders;
  HMTS_DE02_VisualizeDetailedOrder; HMTS_DE03_ProcessOrder.
- The product will be added to a cart in 1 seconds:
  HMTS_CU02_AddProduct; HMTS_CU07_PopAddressAlert;
  HMTS_CU09_VisualizeOrderStatus.

Supportability

- The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed: All use cases.

Implementation

- The implementation of the system is to be done in Java in the Eclipse development environment:
  HMTS_UM01_CreateAccount; HMTS_UM05_LoginTimeOut;
  HMTS_OP01_VisualizeProducts; HMTS_OP02_AddProduct;
  HMTS_OP03_RemoveProduct; HMTS_OP04_VisualizeOrders;
  HMTS_OP05_VisualizeDetailedOrder; HMTS_OP06_ProcessOrder.
- The implementation of the system will be done in Java and Eclipse framework and Android framework:
  HMTS_UM02_Login; HMTS_UM03_ForgetPassword;

HMTS_UM04_Logout; HMTS_UM06_LimitFailedLoginTimes;
HMTS_UM07_RolesControl; HMTS_CU01_VisualizeMenu;
HMTS_CU02_AddProduct; HMTS_CU03_ViewCart;
HMTS_CU04_RemoveProduct; HMTS_CU05_FillOutOrderInfo;
HMTS_CU06_VisualizePOC; HMTS_CU07_PopAddressAlert;
HMTS_CU08_PlaceOrder; HMTS_CU09_VisualizeOrderStatus;
HMTS_DE01_VisualizeOrders; HMTS_DE02_VisualizeDetailedOrder;
HMTS_DE03_ProcessOrder.

4.2. Use Case Diagram

1. Top Level

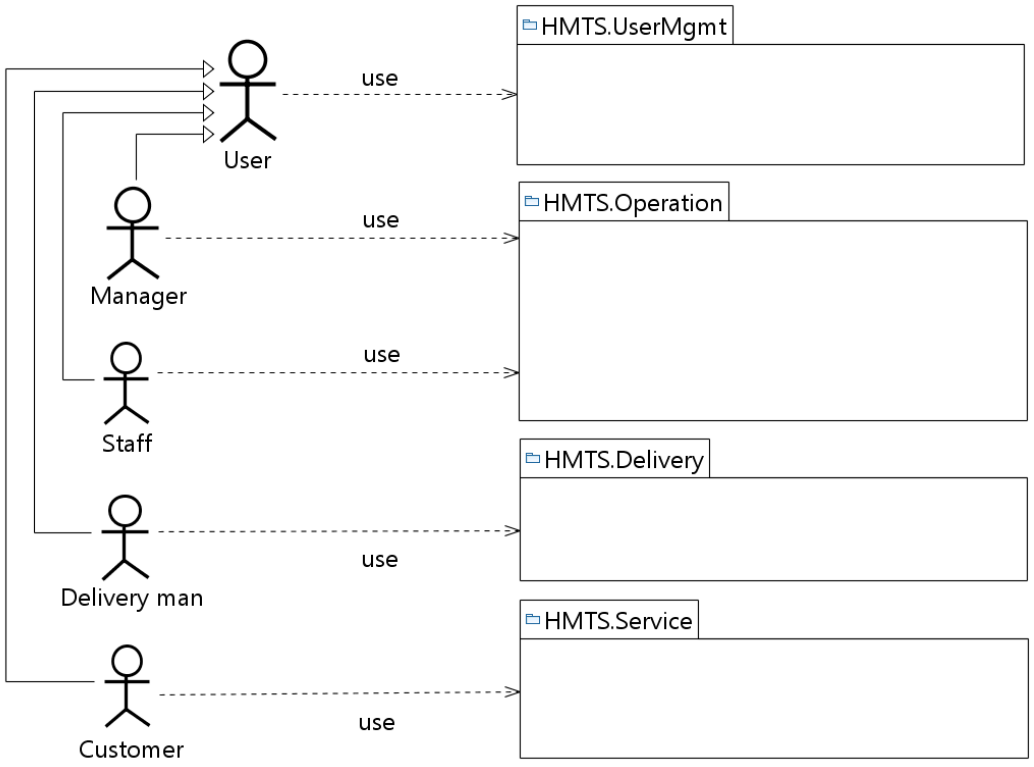


Figure 4.1. The top level of use case diagram.

Figure 4.1 describes several users interacting with the "Happy Meal Take-out System" (from this point on, referred to as HMTS). Each user has a different level of privilege enabling a variety of HMTS functions.

- *User*: Generic entity in storage that encompasses all user roles. Corresponds to a customer, manager, deliveryman, or staff member registered in the system.
- *Staff*: deals strictly with the Operation package functions of HMTS.
- *Manager*: this is a user that deals strictly with the Operation package functions of HMTS. Its role is to oversee the Staff as he has more privileges in the system.
- *Delivery man*: this is a user that deals strictly with the Delivery package functions of HMTS. Its role is to deliver the products ordered to the customer.
- *Customer*: this is a user that deals strictly with the Customer package functions of HMTS. Its role is to purchase products.

2. User Management

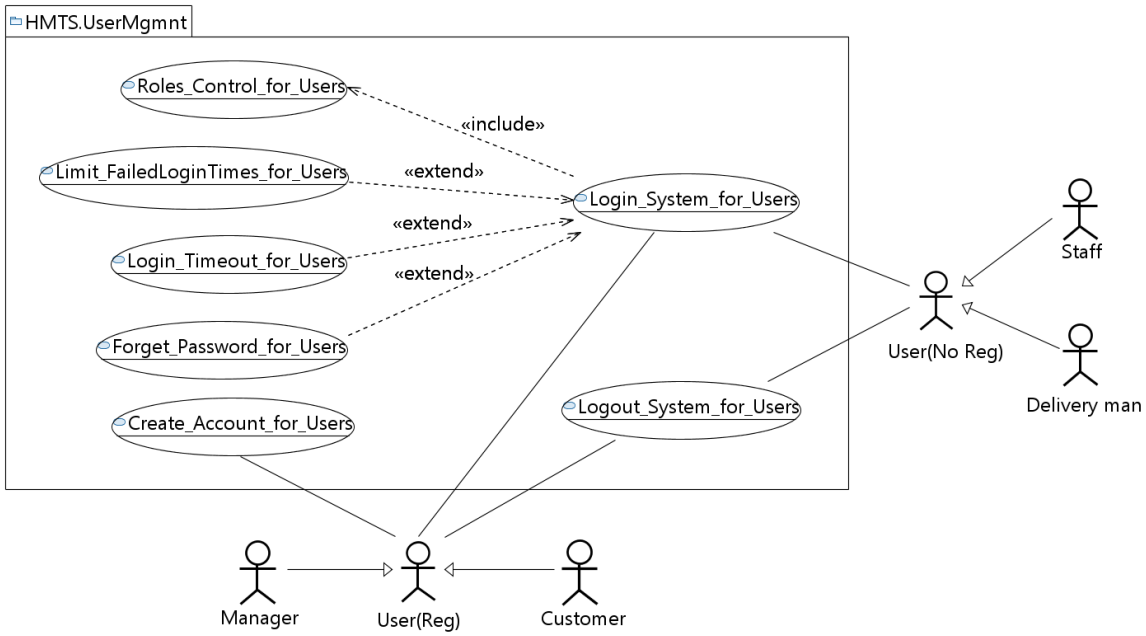


Figure 4.2. Use case diagram for User Management

Figure 4.2 shows all users that can interact with the User Management package. This diagram shows two general roles:

- User (Reg): Actors that are allowed to create a user account.
- User (No Reg): Actors that are not allowed to create a user account.

This package includes the following use cases:

- *Create Account for User*: the system allows to create an account for a user. Users have different roles. There are only two ways to create an account:
  - A customer can create a customer user account.
  - A manager account is provided with the HMTS. The manager can create user accounts with the following roles: Staff and Delivery man.
- *Login System for User*: the system allows the user to log in.
  - *Includes* Roles Control for Users: the system shall use roles control for different roles of users. Different roles have different privileges to the functions in the HMTS.
  - *Extended by Forget Password for User*: this is a security case by which the system helps a user find the account password in a safe way, such as by answering a security question.
  - *Extended by Limit Failed Login Times for User*: this is a security case by which the system shall limit the amount of failed logins for a user.
  - *Extended Login Timeout for User*: this is a security case by which the system times out after a fixed amount of time.
- *Logout System for User*: the system allows users to log out.

According to the roles assigned, each user has access to different HMTS packages. The details are talked in the following sections.

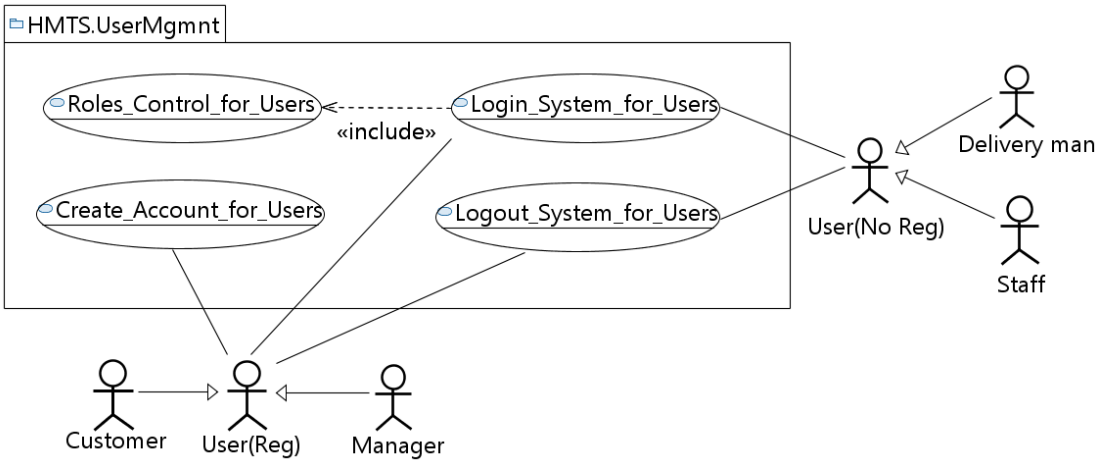


Figure 4.3. Use case diagram for the implementation of User Management.

The uses cases we plan to implement in this package are shown in the Figure 4.3.

3. Operation

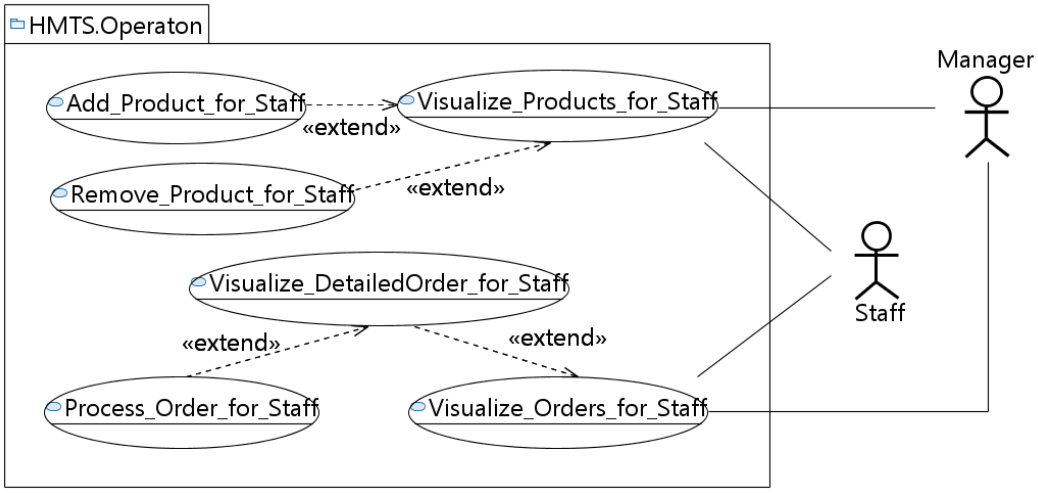


Figure 4.4. Use case diagram for Operation.

Figure 4.4 shows that besides being able to create user accounts for the staff, the Manager has full access to the Operation HMTS package, which includes the following use cases:

- *Visualize Products for Staff*: the system visualizes a product list including dishes, sides, and drinks.
  - *Extended by Add Product for Staff*: the system allows staff to add dishes, sides, or drinks to the product list.
  - *Extended by Remove Product for Staff*: the system allows staff to remove dishes, sides, or drinks from the product list.
- *Visualize Orders for Staff*: the system allows staff to view the list of orders.
  - *Extended by Visualize Detailed Order for Staff*: the system allows staff to visualize detailed order information.
    - *Extended by Process Order for Staff*: the system allows staff to process the order for delivery after the order is paid.

Other users that have access to the Operation HMTS package are the Staff.

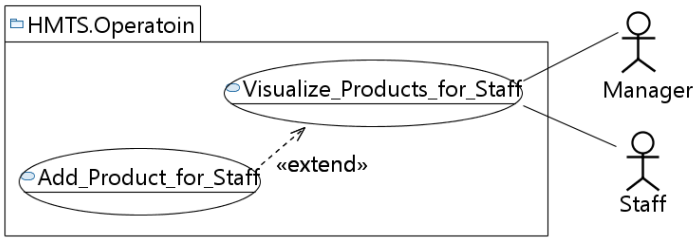


Figure 4.5. Use case diagram for the implementation of Operation.

The uses cases we plan to implement in this package are shown in the figure above.

4. Delivery

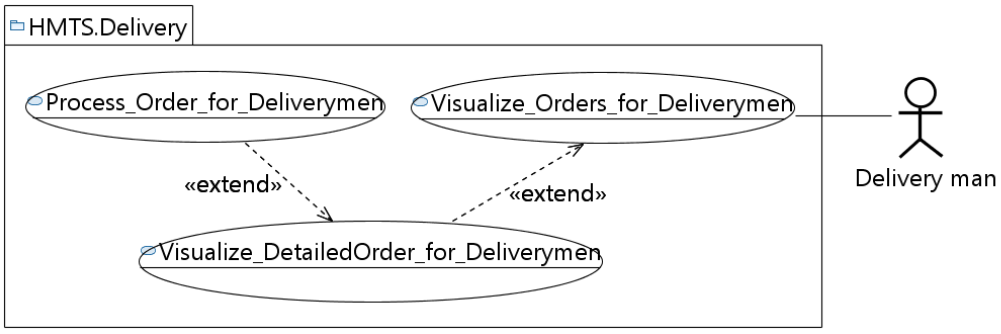


Figure 4.6. Use case diagram for Delivery

The Delivery man has full access to the Delivery package of the HMTS (as shown in Figure 4.6.), which includes the following use cases:

- *Visualize Order for Delivery man*: the system visualizes a list of the orders assigned for delivery.
  - *Extended by Visualize Detailed Order for Delivery man*: the system shall allow the deliveryman to visualize relevant order details.
    - *Extended by Process Order for Delivery man*: the system allows the deliveryman to update the status of an order.

5. Customer

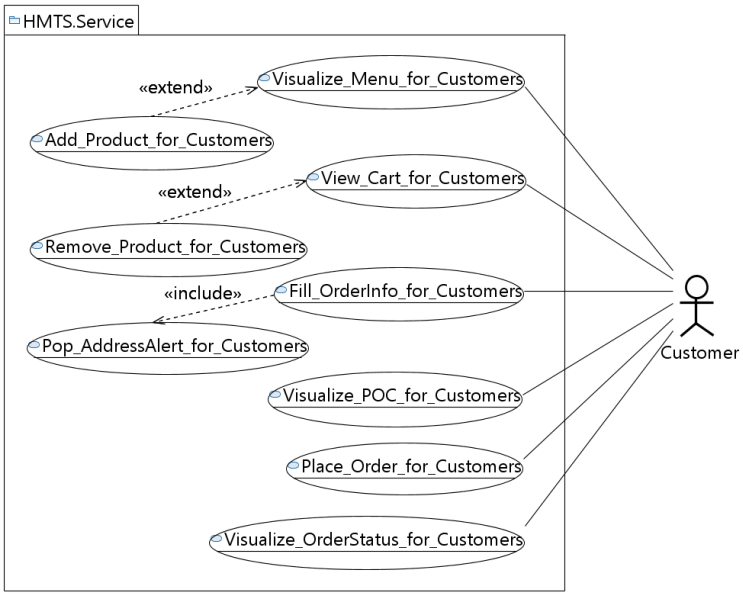


Figure 4.7. Use case diagram for Customer

The Customer only has access to the Customer package of the HMTS (as shown in Figure 4.7.), which includes the following use cases:

- *Visualize Menu for Customer*: the system shall visualize the list of products (dishes, sides, and drinks) in a menu format.
  - *Extended by Add Product for Customer*: the system allows the customer to add dishes, sides, and drinks to a shopping cart.
- *View Cart for Customer*: the system allows the customer to visualize their shopping cart.
  - *Extended by Remove Product for Customer*: the system allows the customer to remove dishes, sides, and drinks from the shopping cart.
- *Fill Order Information for Customer*: the system allows the customer to select a payment method and fill in the corresponding order information.
  - *Includes Pop Address Alert for Customer*: this is a security case by which the system compares the billing and shipping address entered by the customer and pops an alert if they are different, asking the user to edit or confirm that there is no mistake.
- *Place Order for Customer*: the system allows the customer to place an order.

- *Visualize POC (Purchase Order Confirmation) for Customer*: the system visualizes a purchase order confirmation page.
- *Visualize Order Status for Customer*: the system visualizes the status of the order.

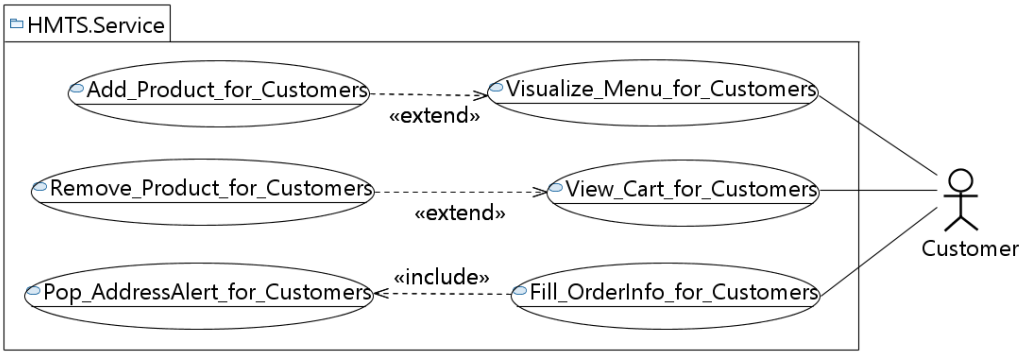


Figure 4.8. Use case diagram for the implementation of Customer.

The uses cases we plan to implement in this package are shown in the Figure 4.8.

4.3. Requirements Analysis

The Requirements Analysis is an important part in the process of the development of the HMTS since its objective is defined a set of approved requirements to meet the needs of the client. This way we can have at the end a formal representation of the needs mentions before and a clear tool to work throughout all the process of the analysis, design and development of the system.

The main goal of the analysis is to produce a model of the system that is correct, complete and verifiable, that way we can ensure that we are meeting the client's needs and expectations. To do this we create 3 different models that complement together the analysis model needed for the system: Functional Model, Analysis Object Model and Dynamic Model. To be able to generate the Functional Model we have to gather all the functional and non-functional requirements with which we were able to create the corresponding scenarios. Once we have this we focus on the next model, the Analysis Object Model. In this model we rely on the scenarios that we defined in the previous model to be able to derive the object diagrams and iteratively, to create a class diagram, identifying at the same time boundary, control and entities objects. Finally, with the help of the Analysis Object Model we were able to define the Dynamic Model which consist of a set of sequences diagrams that modeled the iterations between actors and the objects in the system.

5. Software Architecture

In Chapter 5 we describe the overall design of the Happy Meal Take-out System. In particular, we describe the major subsystems and provide the rationale behind the selection of each architectural pattern. We also describe the mapping of each subsystem in terms of software and hardware with the inclusion of a UML deployment diagram. Finally, we identify persistent data needs through an ER diagram, as well as the security protocols that the HMTS incorporates in its design.

5.1. Overview

In our system there are two types of clients: web and mobile. Furthermore, a database is present. There are multiple user interfaces on both clients, that is, interfaces specifically defined for computers and smartphones. We took functional requirements (see Appendix B) and non-functional requirements into account in designing the architecture of the system. In the non-functional requirements section, performance requires that most processes be completed within 2 seconds. We balance the integration and composition of subsystems and take care of the connection between the subsystems by using two architecture patterns: 4-tier as a primary pattern and MVC as a secondary pattern. This way, the connection between Client and Presentation tier takes less than half a second, access to a database takes less than one second, and other processes take less than half a second. Figure 5.1 shows the primary architecture pattern, 4-tier.

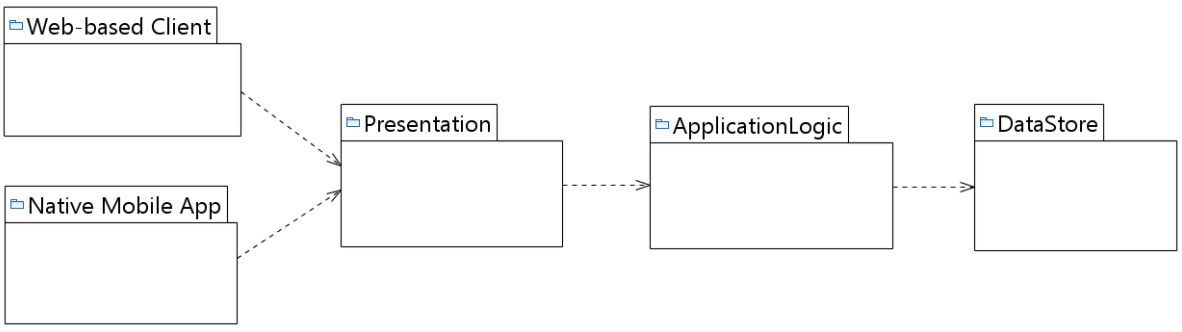


Figure 5.1. The primary architecture pattern, 4-tier.

The client tier represents all device or system clients accessing the. A client can be a Web browser, a mobile application, a Java applet, a WAP phone, a network application, or some

device introduced in the future. It could even be a batch process. In this system, the client tier includes two parts, Computer and Smartphone. For the Computer, there must be browsers on it to access web pages and functionalities through the browsers from the Presentation tier [2]. The other one is a native application deployed on mobile devices. It includes the user interfaces and functions for requesting services through an interface on the Presentation tier. The Smartphone client is used by customers to order meals.

The Presentation tier encapsulates all presentation logic required to service the clients that access the system. The presentation tier intercepts the client requests, conducts session management, controls access to business services, constructs the responses, and delivers the responses to the client. Servlets and JSP reside in this tier. Note that servlets and JSP are not themselves User Interface elements, but they produce UI elements for the website including product management, order management, user management and so on.

The ApplicationLogic tier provides the business services required by the Presentation tier. The tier contains the business data and business logic. Typically, most business processing for the application is centralized into this tier. It is possible that, due to legacy systems, some business processing may occur in the resource tier. Enterprise bean components are the usual and preferred solution for implementing the business objects in the business tier.

The DataStore tier is responsible for communicating with external resources and systems such as data sources and legacy applications. The business tier is decoupled with the DataStore tier whenever the business objects require data or services that reside in the resource tier. The components in this tier can use JDBC, MySql connector technology, and MyBatis middleware to work with the data sources. This is the tier that contains the business entity data and external resources [3].

One advantage of using the 4-tier architecture pattern is the ability to carry out independent tier upgrading and changing without affecting other tiers. In an object-oriented world, interface-dependency implementation can decouple all layers very well so that each layer can change individually without significantly affecting other layers. Interface-dependency implies a layer depends on another layer by interfaces only, rather than by concrete classes. Furthermore, the dependency of a layer only on its directly-below layer minimizes the side effect of a layer change on the whole system [1]. For example, if we keep the interfaces unchanged, we can

update or replace the implementation of any layer independently without affecting the whole system. Due to the fluctuation of business requirement and technology, radically changing the implementation of a layer is a frequent occurrence.

Another advantage of the 4-tier architecture is its friendliness and efficiency for development. The decoupled layers are logic software component grouped mainly by functionality; they are very software development-friendly and efficient. Each layer can be assigned individually to a team who specializes in their functional area, thus handling the relevant task better and more efficiently.

In the Smartphone client, the secondary pattern, MVC, is used. See Figure 2.2 below.

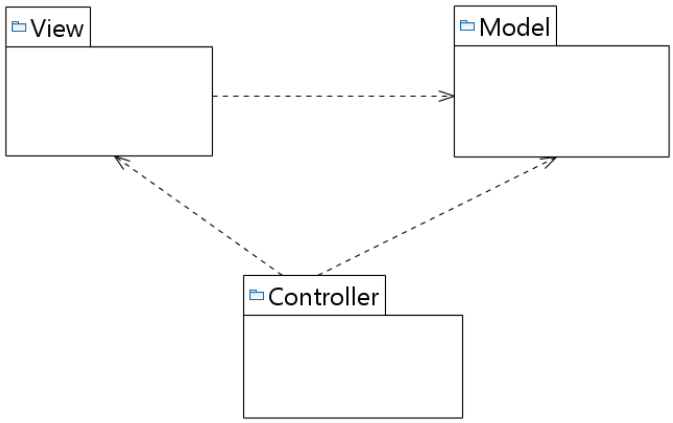


Figure 5.2. The secondary architecture pattern MVC.

One of the advantages of the MVC pattern is multiple view support. Due to the separation of the model from the view, the user interface can display multiple views of the same data at the same time.

Another advantage afforded by the MVC architecture is change accommodation. User interfaces tend to change more frequently than business rules. Different colors, fonts, screen layouts, and levels of support for new devices such as cell phones are concrete examples of the above. Because the model does not depend on the views, adding new types of views to the system generally does not affect the model. As a result, the scope of change is confined to the view.

5.2. Subsystem Decomposition

In this section we describe each of the major subsystems in detail, identifying the use cases associated with each subsystem. For use case diagrams, see Section 4.2.

1. Web-based Client

This is the web client subsystem. It is deployed through the Presentation subsystem, which produces UI elements for the website through servlets and JSP in order to enable the user to carry out all functionalities proper to the given user role. Customers and staff members can create accounts in the HMTS (HMTS_UM01_CreateAccount), but for security reasons only staff members can upgrade the user roles (HMTS_UM07_RolesControl). All system users with a valid account can log in and out of the HMTS (HMTS_UM02_Login, HMTS_UM04_Logout). Customers can visualize the menu (HMTS_CU01_VisualizeMenu) but staff members have higher authority that enables them to alter the menu by visualizing details on the products (HMTS_OP01_VisualizeProducts) and adding new products (HMTS_OP02_AddProduct). Customers have the ability to add products to a shopping cart (HMTS_CU02_AddProduct), visualize the cart (HMTS_CU03_ViewCart) and edit its contents by removing previously added products (HMTS_CU04_RemoveProducts). The customer can check out by selecting a payment method and providing all the required information (HMTS_CU05_FillOutOrderInfo). For security reasons, if there was an error in the transaction he or she will be alerted (HMTS_CU07_PopAddressAlert) and will be able to go back and modify the information in order to check out again.

2. Native Mobile Application

This is the mobile client subsystem. Although it requests services through an interface on the Presentation subsystem, it incorporates its own user interface built on an MVC architecture. Its key components are the HttpService Model, the Activity View, and the MobileController Controller. This subsystem enables the customer to register (HMTS_UM01_CreateAccount), log in (HMTS_UM02_Login), and visualize the menu (HMTS_CU01_VisualizeMenu). It also allows the customer to complete an order transaction, going through the steps of selecting one or multiple items from the menu and placing them on a shopping cart (HMTS_CU02_AddProduct), visualizing the cart (HMTS_CU03_ViewCart) and possibly removing one or more previously chosen items (HMTS_CU04_RemoveProduct), checking out

by selecting whether to pay with cash or credit card and filling the required fields (HMTS_CU05_FillOutOrderInfo). For security reasons, if there was an error, the subsystem visualizes an alert (HMTS_CU07_PopAddressAlert) and the customer can update the payment information and re-submit the order. The user can log out when he has completed his interaction with the HMTS (HMTS_UM04_Logout).

3. Presentation

This subsystem is in charge mediating the service of the mobile and computer client subsystems and the higher-level HMTS subsystems. Its functionality intercepts all client requests distinguishing between mobile and web requests and filtering them accordingly. It is capable of querying the other subsystems (ApplicationLogic and DataStore) and to construct and deliver the appropriate responses to the clients. This subsystem is also in charge of producing UI elements for the computer subsystem since it incorporates servlets and JSP. More specifically, the Presentation subsystem provides the logic and interfaces for creating an account (HMTS_UM01_CreateAccount), logging in (HMTS_UM02_Login), visualizing the menu (HMTS_CU01_VisualizeMenu), visualize menu products (HMTS_OP01_VisualizeProducts), adding products to the menu (HMTS_OP02_AddProduct), visualizing a shopping cart (HMTS_CU03_ViewCart), adding and removing items from the cart (HMTS_CU02_AddProduct, HMTS_CU04_RemoveProduct), and checking out (HMTS_CU05_FillOutOrderInfo). Security is also supported, as the subsystem provides the logic for modifying the user role (HMTS_UM07_RolesControl) and displaying alerts when inconsistent information is submitted by the customer (HMTS_CU07_PopAddressAlert).

4. ApplicationLogic

This subsystem is at the core of the HMTS. It incorporates and provides all the business services that the Presentation subsystem requires in order to meet the needs of the two client subsystems. It incorporates functionality to direct, build, register, service, and control. It is the tier at which high-level functionality resides, including the ability to perform persistent data management in connection with the DataStore subsystem. Its main interface to the Presentation subsystem is through its main controller via a DataTransfer function. In this way, the ApplicationLogic subsystem enables creating an account (HMTS_UM01_CreateAccount),

modifying the user role (HMTS_UM07_RolesControl), logging in (HMTS_UM02_Login), visualizing the menu (HMTS_CU01_VisualizeMenu), visualize menu products (HMTS_OP01_VisualizeProducts), adding products to the menu (HMTS_OP02_AddProduct), visualizing a shopping cart (HMTS_CU03_ViewCart), adding and removing items from the cart (HMTS_CU02_AddProduct, HMTS_CU04_RemoveProduct), checking out (HMTS_CU05_FillOutOrderInfo), and displaying alerts when inconsistent information is submitted by the customer (HMTS_CU07_PopAddressAlert).

5. DataStore

This is the last subsystem of our 4-tier architecture. It incorporates persistent data management through the usage of JDBC, MySQL connectors, MyBatis middleware, etc. in order to interact with data sources and other external resources. Linked to the rest of the HMTS through its DataMapper function, the subsystem is capable of instantiating SQL sessions, entities, objects, and to enable the HMTS to select, insert, select, update, and delete persistent data. It is by relying on the DataStore subsystem that the HMTS is capable of carrying out its core tasks of enables creating an account (HMTS_UM01_CreateAccount), modifying the user role (HMTS_UM07_RolesControl), logging in (HMTS_UM02_Login), visualizing the menu (HMTS_CU01_VisualizeMenu), visualize menu products (HMTS_OP01_VisualizeProducts), adding products to the menu (HMTS_OP02_AddProduct), visualizing a shopping cart (HMTS_CU03_ViewCart), adding and removing items from the cart (HMTS_CU02_AddProduct, HMTS_CU04_RemoveProduct), checking out (HMTS_CU05_FillOutOrderInfo), and displaying alerts when inconsistent information is submitted by the customer (HMTS_CU07_PopAddressAlert).

5.3. Hardware and Software Mapping

In this section, we describe the mapping of each subsystem to hardware and software, including a UML deployment diagram. See Figure 5.3 below.

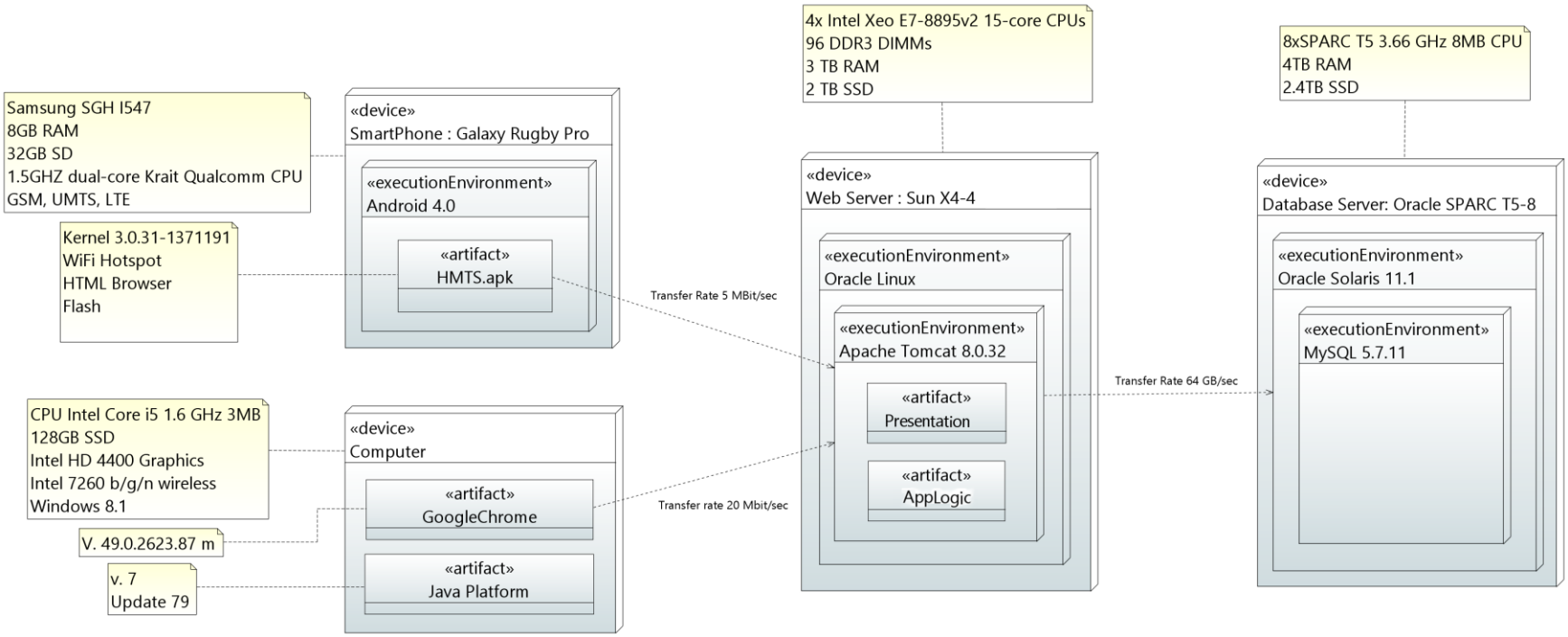


Figure 5.3. Deployment diagram for HMTS system.

1. **Clients**

The HMTS supports two clients: Web-based Client and Native Mobile Application. The former is based on a web interface. The latter is based on a mobile interface implementation.

- ***Web-based Client***

  In terms of hardware, the web-based client is deployed on a PC or laptop with a CPU Intel Core i-5-4200 1.6 GHz 3 MB, a 128GB SSD hard drive, Intel HD 4400 graphics, and an Intel 7260 b/g/n wireless network card. As for the software requirements, the computer runs the Windows operating system 8.1, have the Java Platform Version 7 Update 79, and have the Google Chrome browser software version 49.0.2623.87m installed. It is capable of communicating with the higher tiers at a minimum transfer rate of 20 Mbit/sec.

- ***Native Mobile Application***

  In terms of hardware, the native mobile application is deployed on an Android mobile device Samsung SGH-I547 touchscreen with 8 GB internal memory, 32GB external memory, 16M AMOLED display with 800x480 pixel resolution, 1.5GHZ Dual-core Krait Qualcomm CPU, and network support for GSM Quad-band, UMTS tri-band, and LTE dual-band. Software-wise, the smartphone includes the Android 4.0 Ice Cream Sandwich OS or higher, a kernel version 3.0.31-1371191 or higher, WiFi Hotspot technology, HTML browser, Flash, and the HMTS.apk application file deployed. It is capable of communicating with the higher tiers at a minimum transfer rate of 5 Mbit/sec.

2. **Web Server**

The HMTS web server incorporates two tiers, the Presentation and the ApplicationLogic. Hardware-wise, the server is an Oracle Sun Server X4-4 x86 with 4 Intel Xeon E7-8895 v2 microprocessors with 15 cores each, 96 DDR3 DIMMS providing up to 3 TB RAM, and 2 TB of internal SSD storage. I/O throughput is up to 64 GB/sec. In terms of software, the web server runs Oracle Linux and Apache Tomcat 8.0.32. Servlets and JSP are present in order to deploy HTML user interfaces, and to handle and process requests from both clients.

3. **Database Server**

   The HMTS database server is part of the last tier, DataStore. Hardware-wise, the server is an Oracle SPARC T5-8 with 8 SPARC T5 CPUs 3.6GHz 8 MB, 4 TB RAM and 2.4 TB internal memory. I/O throughput is up to 258 GB/sec. In terms of software, the web server runs Oracle Solaris 11.1 or above and it includes Oracle VM Server for SPARC 3.0, as well as the Oracle Solaris ZFS default file system. JDBC 4.2 Java API, MySQL 5.7.11 RDBMS, and MyBatis 3.3.0 middleware are incorporated in order to enable and support the communication with external data sources and legacy applications.

5.4. Persistent Data Management

The system contains multiple persistent sets of data and uses a relational database management system, MySQL 5.7.11, to manage the data.

MySQL is a relational database management system (RDBMS) which has more than 11 million installations, and is owned by Oracle. MySQL is a fast, stable, robust, easy to use, and true multi-user, multi-threaded SQL database server.

MySQL provides a modeling tool to help users to create an E-R diagram. Figure 5.4 below shows an Entity Relationship diagram for the system.

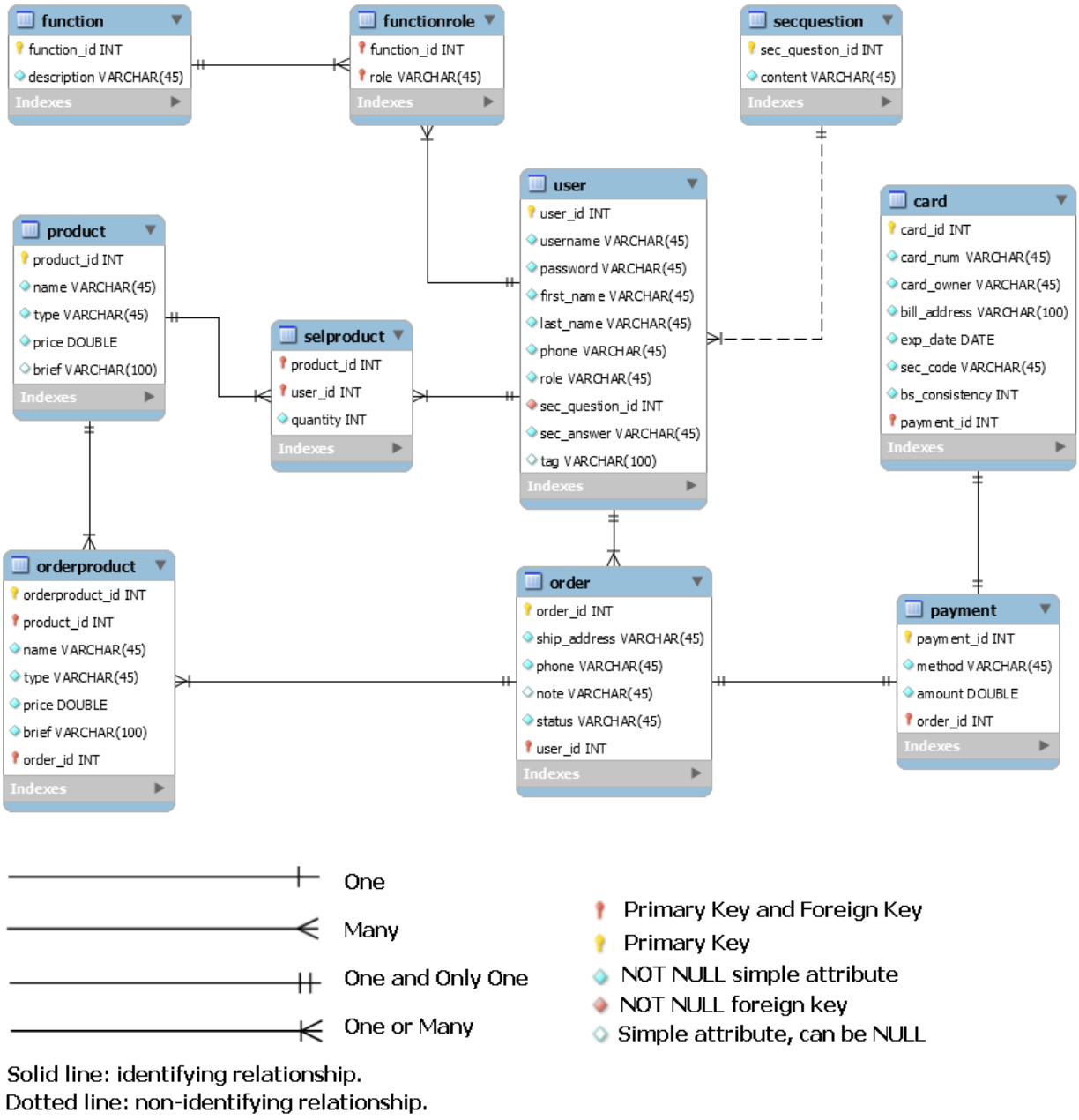


Figure 5.4. E-R diagram of the system.

User entity: used to store information of user accounts that include roles of Manager, Staff, Deliveryman and Customer.

SecQuestion entity: used to store security questions.

Function entity: used to store specification of high-level functionalities.

FunctionRole entity: used to store relationships between functionalities and roles.

Product entity: used to store information of products offered by the restaurant..

SelProduct entity: used to store information of products in customer's shopping cart.

OrderProduct entity: used to store information of products in orders.

Order entity: used to store information of orders.

Payment entity: used to store information of payments made by customers.

Card entity: used to store information of credit and debit cards during transaction.

5.5. Security Management

Table 5.1 highlights the read-write capabilities of each Actor as it pertains to the persistent data. The table shows which system roles have access to specific persistent data entries. This table is meant to aid in system understanding and in grasping the privileges of different user roles. It also ensures consistency in implementation.

Table 5.1. Persistent data vs. System Roles.

		System Roles			
		Customer	Manager	Staff	Delivery-man
Persistent Data Entities	function		✔	✔	✔*
	functionrole		✔	✔*	✔*
	product	✔	✔	✔	
	orderproduct	✔*	✔	✔	✔*
	selproduct	✔*	✔	✔	
	user	✔*	✔	✔	✔*
	order	✔*	✔	✔	✔*
	secquestion	✔	✔	✔	✔
	card	✔*			
	payment	✔*	✔	✔	✔*

Note: ✔ - read/write abilities; ✔ - read only ability; * denotes access to only selected tuples

Table 5.2 below highlights the System Roles as they pertain to the Use Cases described in this document. The use cases described here represent all of the use cases within the system, and this table simply highlights the common threads about use cases as it pertains to which specific roles the use cases refer to. This greatly aids in understanding user privileges based on user roles.

Table 5.2. Use Cases vs. System Roles.

		System Roles			
		Customer	**Manager**	**Staff**	**Deliveryman**
Use Case Description	CU01_VisualizeMenu*	**Yes**	No	No	No
	CU02_AddProduct*	**Yes**	No	No	No
	CU03_ViewCart*	**Yes**	No	No	No
	CU04_RemoveProduct*	**Yes**	No	No	No
	CU05_FillOutOrderInfo*	**Yes**	No	No	No
	CU06_VisualizePOC	**Yes**	No	No	No
	CU07_PopAddressAlert*	**Yes**	No	No	No
	CU08_PlaceOrder	**Yes**	No	No	No
	CU09_VisualizeOrder	**Yes**	No	No	No
	DE01_VisualizeOrder	No	No	No	**Yes**
	DE02_VisualizeDetails	No	No	No	**Yes**
	DE03_ProcessOrder	No	No	No	**Yes**
	OP01_VisualizeProducts*	No	**Yes**	**Yes**	No
	OP02_AddProduct*	No	**Yes**	**Yes**	No
	OP03_RemoveProduct	No	**Yes**	**Yes**	No
	OP04_VisualizeOrders	No	**Yes**	**Yes**	No
	OP05_VisualizeDetails	No	**Yes**	**Yes**	No
	OP06_ProcessOrder	No	**Yes**	**Yes**	No
	UM01_CreateAccount*	**Yes**	**Yes**	No	No
	UM02_Login*	**Yes**	**Yes**	**Yes**	**Yes**
	UM03_ForgetPassword	**Yes**	**Yes**	**Yes**	**Yes**
	UM04_Logout*	**Yes**	**Yes**	**Yes**	**Yes**
	UM05_Timeout	**Yes**	**Yes**	**Yes**	**Yes**
	UM06_LimitFailedLoginTimes	**Yes**	**Yes**	**Yes**	**Yes**
	UM07_RolesControl*	**Yes**	**Yes**	**Yes**	**Yes**
Implemented use cases are marked with an asterisk (*) and can be found in Appendix B.					

6. Detailed Design

In this chapter we first give an overview of the system by introducing the minimal class diagram for the subsystems. Then we introduce the state machines, including a state machine for the overall system and one state machine for each subsystem. After that, we show the object interactions in our system by giving the refined sequence diagrams. Finally we talk about detailed class design, including the introduction of each class, and OCL for the control object in each major subsystem.

6.1. Overview

In Figure 6.1, we present the classes of the mobile client subsystem. *HttpService* manages data, logic, and rules. *MobileController* handles authentication, session management, and requests. *SplashActivity* displays subsystem description information. *LoginAcivity* oversees login functionality. *RegisterActivity* oversees user registration. *MenuActivity* oversees everything related with the menu. *CartActivity* oversees everything related to a user's shopping cart. *PaymentActivity* oversees the selection of a payment method. *FillOrderInfoActivity* provides check-out and order placement functionality. *CardActivity* oversees secure processing of credit or debit card.

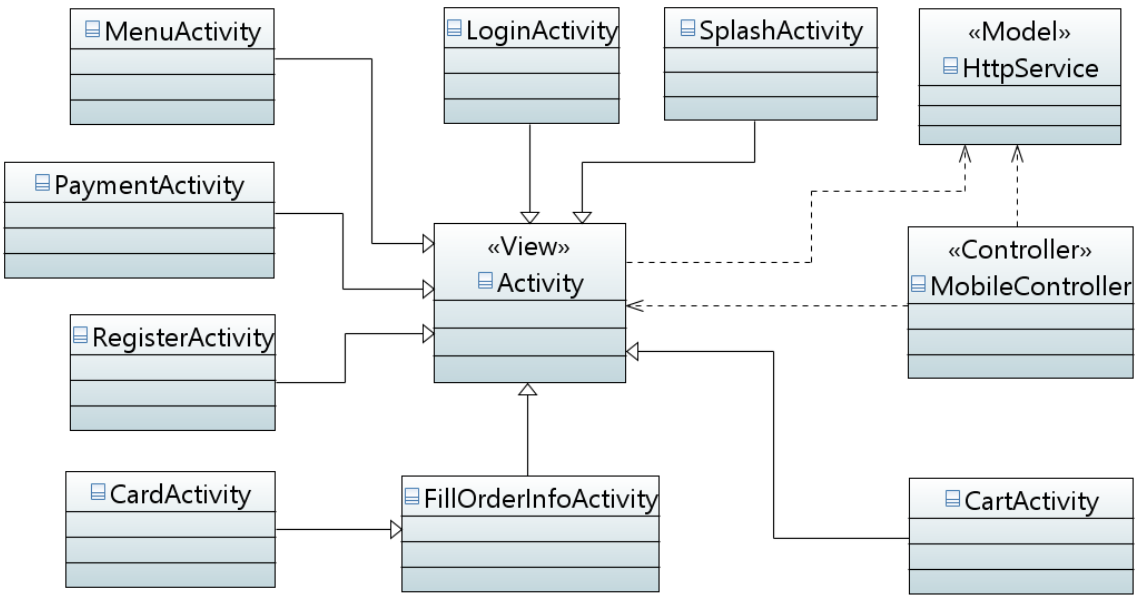


Figure 6.1. Minimal class diagram for mobile client subsystem.

In Figure 6.2, we present the classes of the Presentation subsystem. Page produces the user interface elements. HomePage provides generalized visualization of available HMTS functionality. MenuForm provides visualization of the menu. LoginPage oversees authentication. RegistrationPage enables user registration. ProductsPage enables visualization and interaction with products on the menu. AddProductPage enables addition of new products to menu. MobileInterface handles the logic of the mobile client subsystem, which provides its own user interface. Data and FrontController handle data exchange.

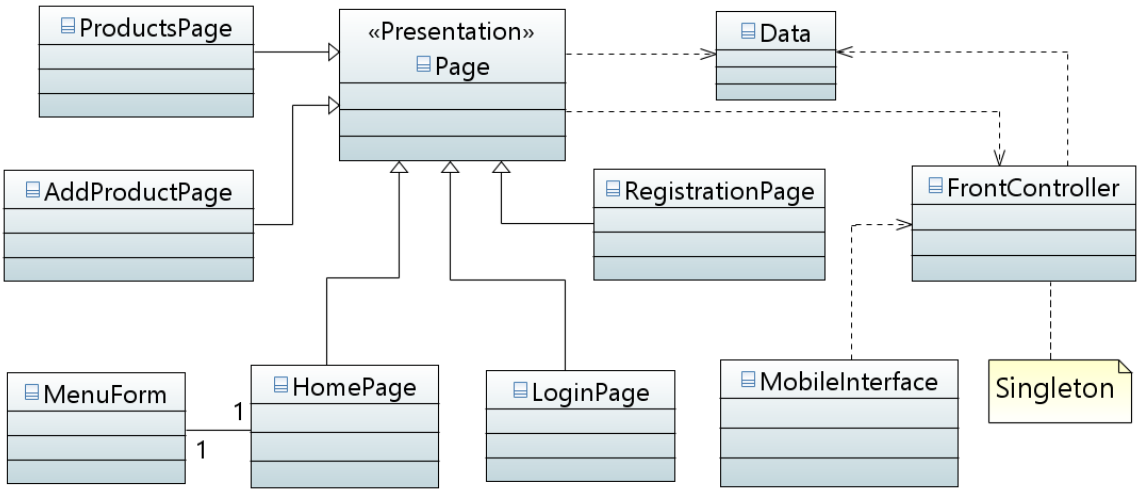


Figure 6.2. Minimal class diagram for presentation subsystem.

In Figure 6.3, we present the classes of the *ApplicationLogic* subsystem. *ServiceService* enables processing service requests. *ServiceBuilder* handles the construction of service requests. *IBuilder* handles obtaining data for service requests. *Director* oversees the construction and processing of service requests. *OerationBuilder* handles the construction of operation requests. *OperationService* oversees the processing of operation requests. *UserBuilder* oversees the creation of user requests for session management. *UserService* handles session management. *Strategy* oversees the registration of new users based on client type. *WebRegister* and *MobileRegister* handle registration requests based on their respective client type. *Controller* is the core processor of the subsystems, providing control and coordination. *OperationController* services all operation requests. *ServiceController* services all service requests. *UserController* services all session management requests. *DataTransfer* intercepts all requests. *ShoppingCart* and *Menu* oversee the creation of the menu and of the shopping carts.

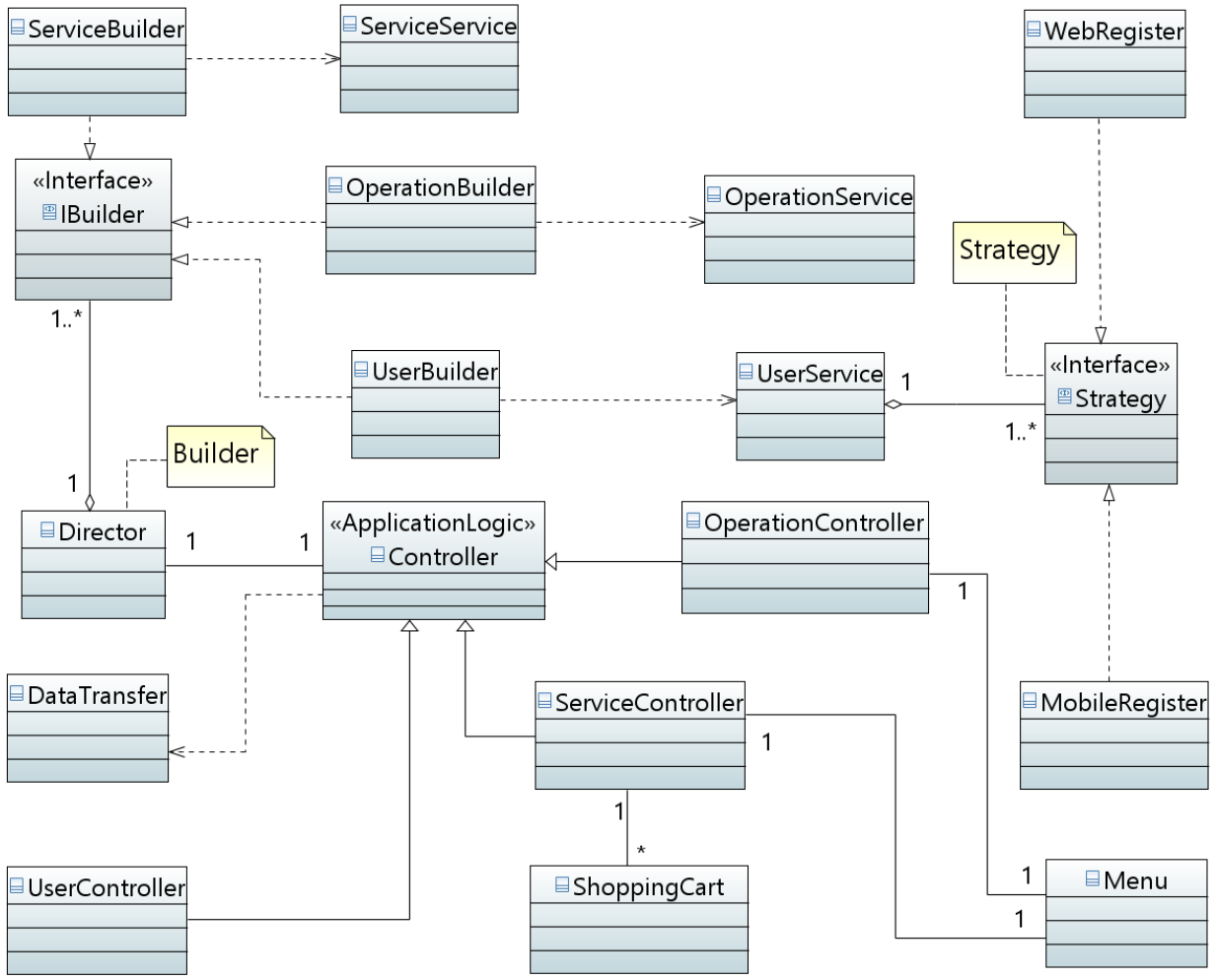


Figure 6.3. Minimal class diagram for application logic subsystem.

In Figure 6.4, we present the classes of the *DataStore* subsystem. *DataMapper* oversees the interaction with data. *SqlSessionFactory* initiates SQL sessions. *SqlSessionManager* manages SQL sessions. *SqlSessionTemplate* provides structure for SQL sessions. *SqlSession* controls SQL sessions. *Function* enables global class functions. User oversees user accounts. *SecQuestion* oversees user account security. *FunctionRole* handles user privileges. *Product* oversees items in the menu. *SelProduct* enables interaction with products. *Order* oversees the creation of orders. *OrderProduct* oversees details within the orders. *Payment* oversees monetary transactions. *Card* oversees credit and debit card monetary transactions.

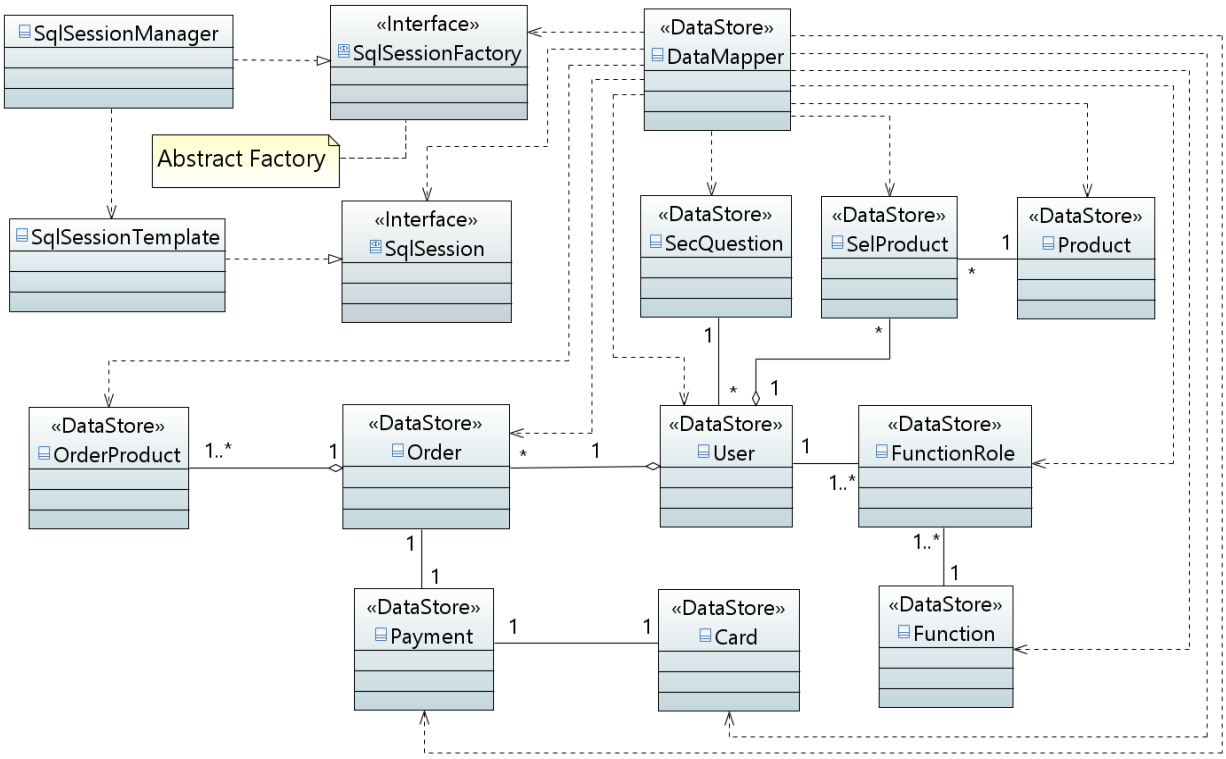


Figure 6.4. Minimal class diagram for data store subsystem.

For the detailed class diagrams, see Appendix D. These describe the classes for each subsystem with four design patterns.

6.2. State machine model

In this section, we introduce the state machines of our system, including a state machine for the overall system and one state machine for each subsystem.

1. Top Level

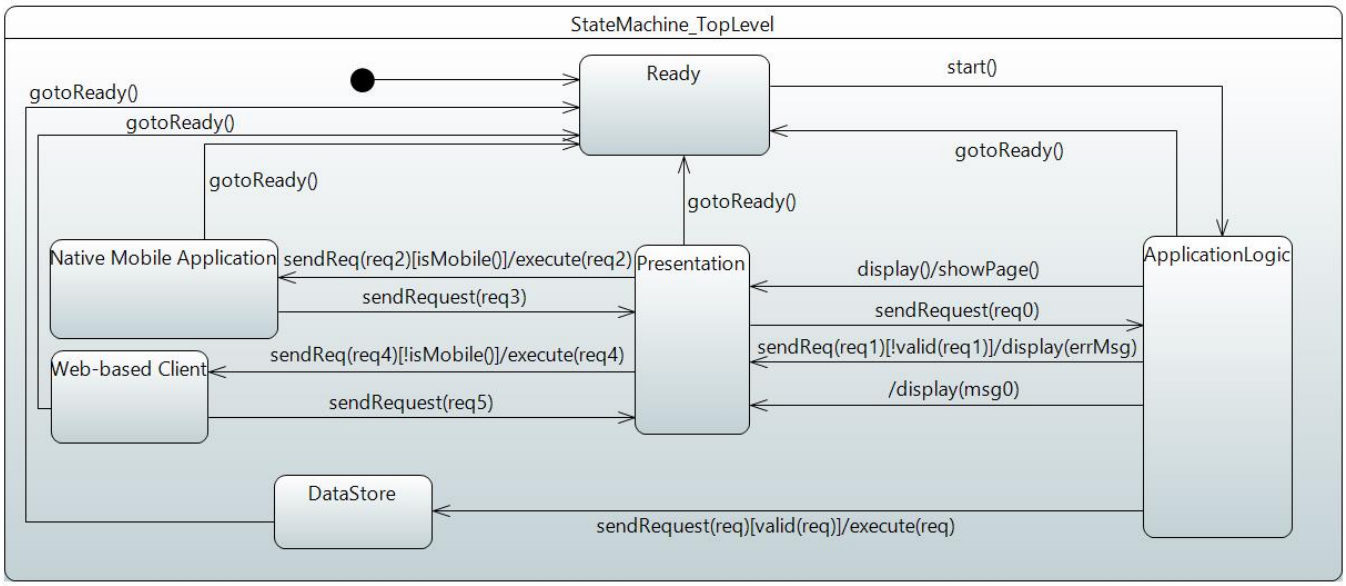


Figure 6.5. State Machine – Top Level

In the top level state machine (see Figure 6.5), we have 6 states:

- **Ready**: the system is ready to launch.
- **Native Mobile Application:** this state represents the mobile client subsystem and is further expanded to a sub state machine.
- **Web-based Client:** this state represents the browser client subsystem and is further expanded to a sub state machine.
- **Presentation:** this state represents the presentation subsystem and is further expanded to a sub state machine.
- **ApplicationLogic:** this state represents the application logic (process) subsystem and is further expanded to a sub state machine.
- **DataStore**: this state represents the data store subsystem and is further expanded to a sub state machine.

2. Native Mobile Application

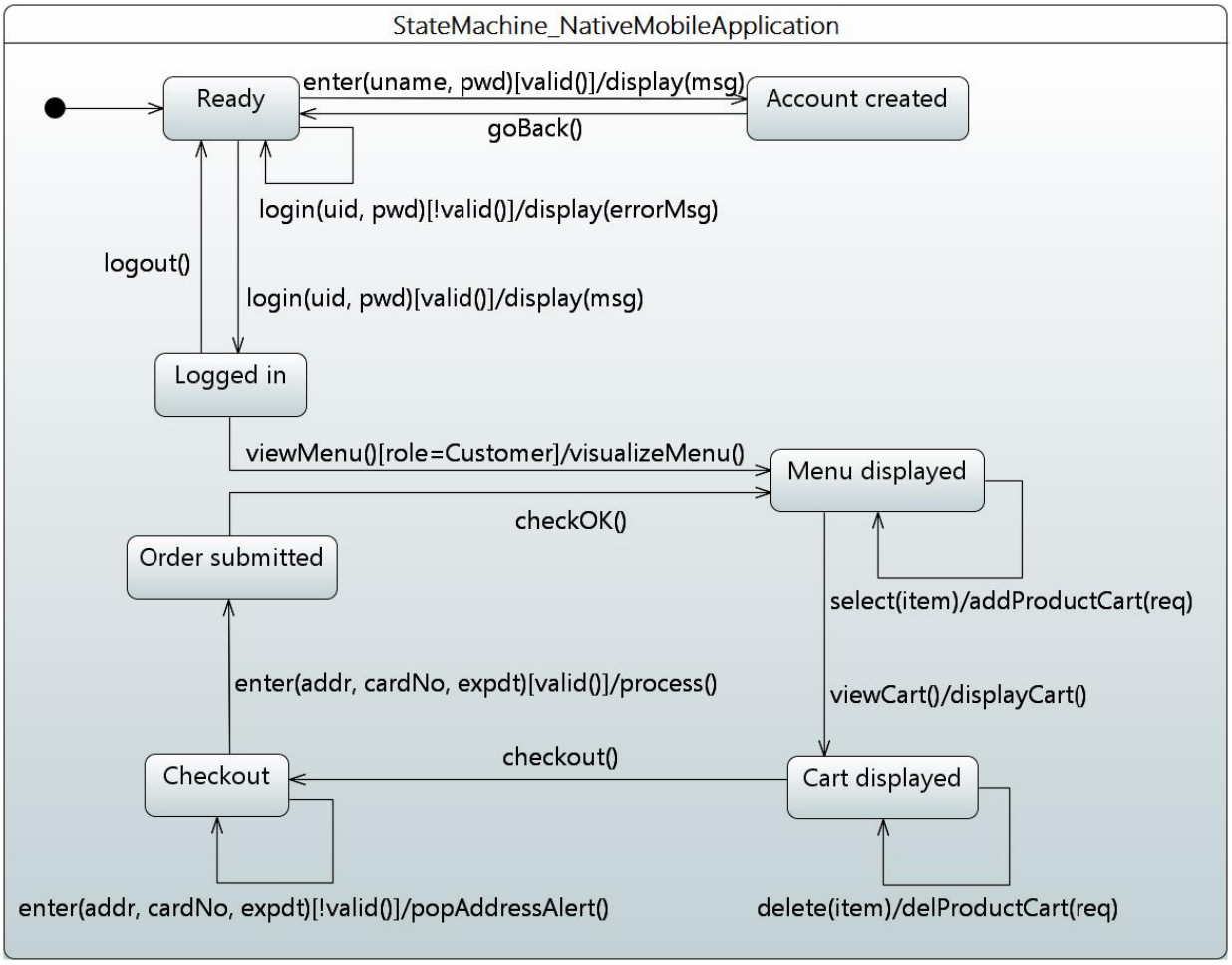


Figure 6.6. State Machine – Native Mobile Application

In the Native Mobile Application state machine (see Figure 6.6), we have 7 states:

- **Ready**: the system is ready to use.
- **Account created:** the customer has successfully registered a user account.
- **Logged in:** the actor has logged in the system.
- **Menu displayed:** the menu is displayed, which contains all the products for the user to choose from.
- **Cart displayed:** the user's shopping cart is displayed.
- **Checkout:** the forms are displayed for the user to enter payment information.
- **Order submitted:** the checkout process is finished and the order is submitted successfully.

3. Web-based Client

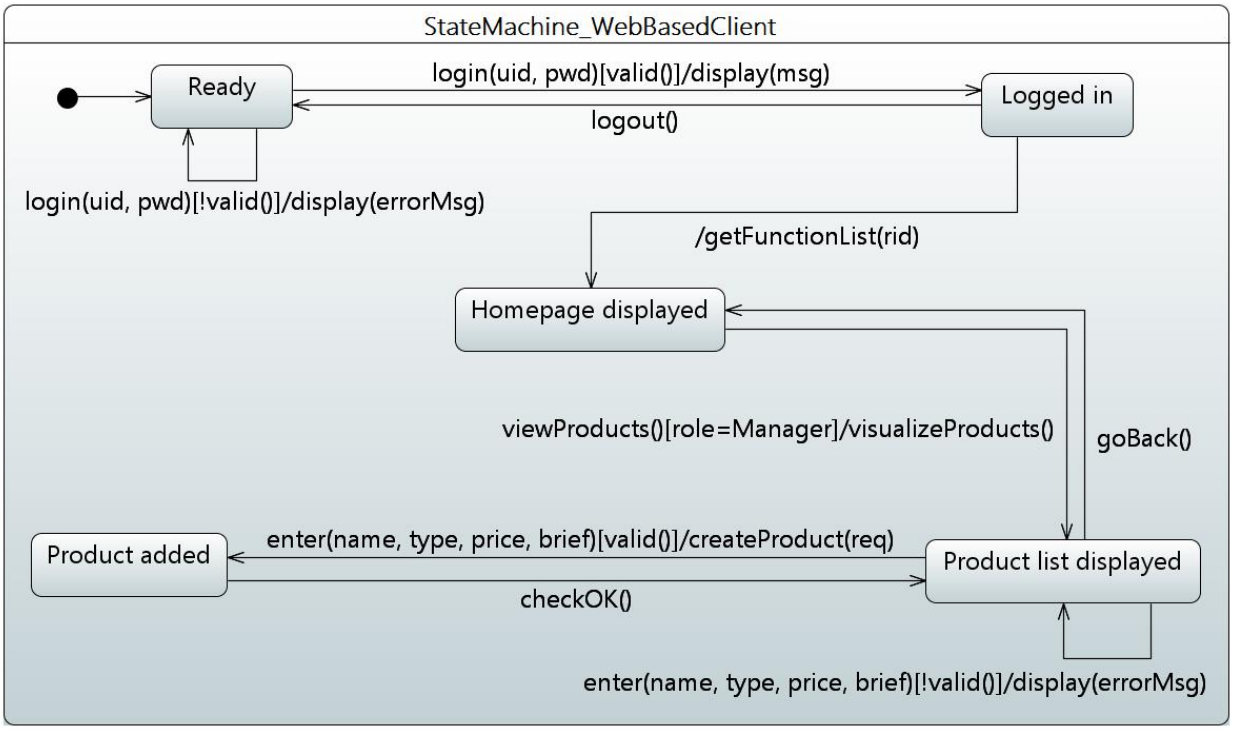


Figure 6.7. State Machine – Web-based Client

In the Web-based Client state machine (see Figure 6.7), we have 5 states:

- **Ready**: the system is ready to use.
- **Logged in:** the user has logged in the system.
- **Homepage displayed:** the homepage is displayed, which contains the functions the logged-in user can access.
- **Product list displayed:** the list of all the products is displayed.
- **Product added:** a new product is added by the user to the shopping cart.

4. Presentation

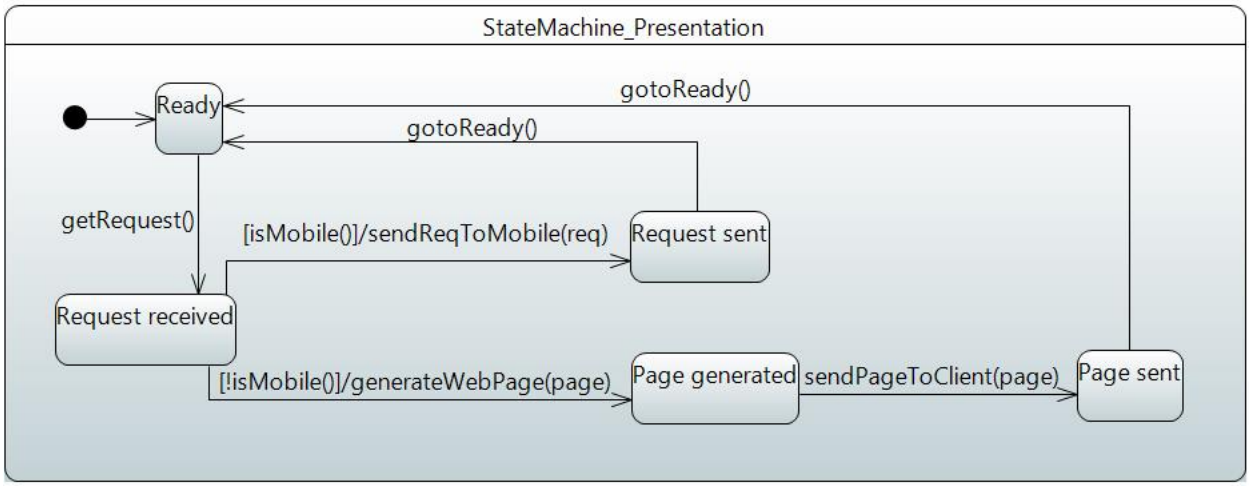


Figure 6.8. State Machine – Presentation

In the Presentation state machine (see Figure 6.8), we have 5 states:

- **Ready**: the system is ready to use.
- **Request received:** a request is received, which will be handled differently depending on whether it is sent to mobile or web browser.
- **Request sent:** the request is sent to the mobile.
- **Page generated:** a web page is generated according to the request.
- **Page sent:** the web page is sent to the web browser.

5. Application Logic

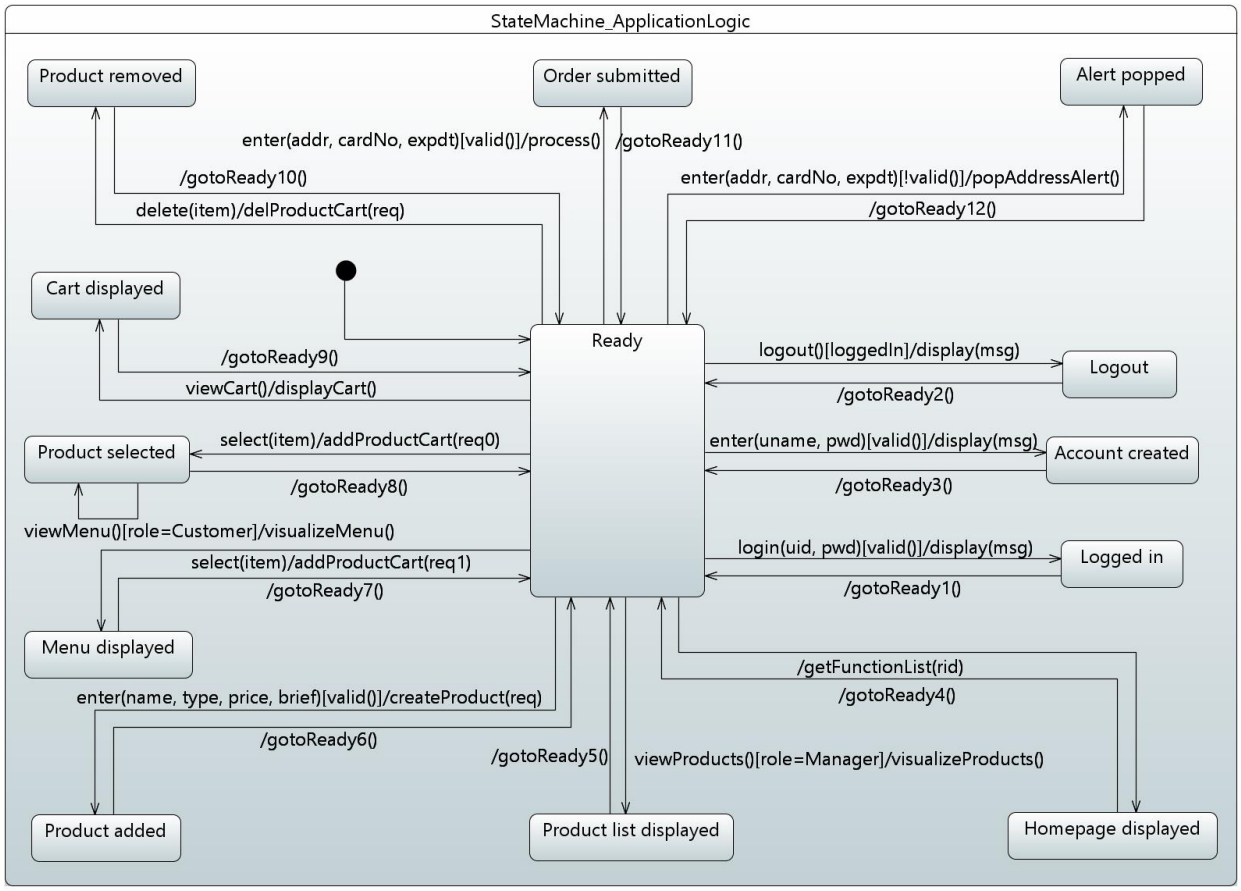


Figure 6.9. State Machine – Application Logic

In the Application Logic state machine (see Figure 6.9), we have 13 states:

- **Ready**: the system is ready to use.
- **Logged in:** the user has logged in the system.
- **Account created:** the customer has successfully registered a user account.
- **Homepage displayed:** the homepage is displayed, which contains the functions the logged-in user can access.
- **Product list displayed:** the list of all the products is displayed.
- **Product added:** a new product is added by manager or staff to the menu.
- **Menu displayed:** the menu is displayed, which contains all the products for the user to choose from.
- **Product selected:** one product is selected by the user, which is added to the cart.
- **Cart displayed:** the user's shopping cart is displayed.
- **Product removed:** one product is removed from the cart by the user.

- **Order submitted:** the checkout process is finished and the order is submitted.
- **Alert popped:** an alert is visualized the billing address and the shipping address differ.
- **Logout:** the user has logged out of the system.

6. Data Store

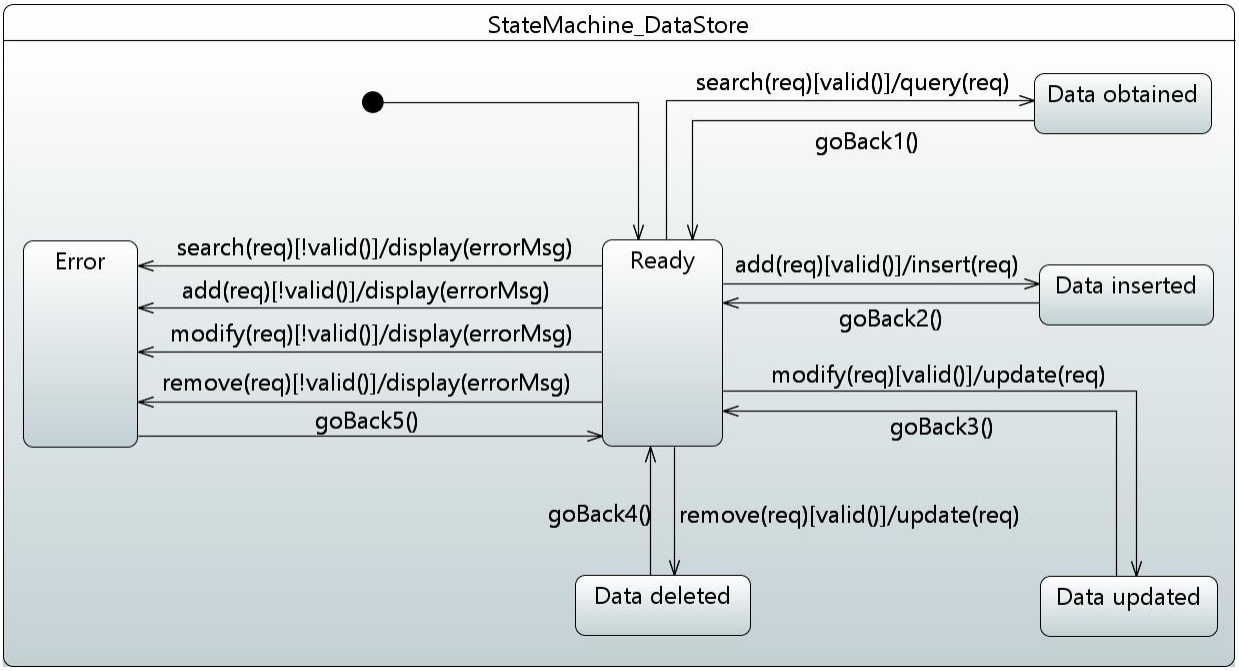


Figure 6.10. State Machine – Data Store

In the Data Store state machine (See Figure 6.10), we have totally 6 states:

- **Ready**: the system is ready to use.
- **Data obtained:** a query operation is executed and the result is obtained from the database.
- **Data inserted:** an insert operation is executed and the data is inserted into the database.
- **Data updated:** an update operation is executed and the data in the database is updated.
- **Data deleted:** a delete operation is executed and the data is deleted from the database.
- **Error**: the system detected an error and deployed an error message. The operation is not executed successfully.

6.3. Object Interaction

In this section we present a refinement of our sequence diagrams showing the interactions between objects.

1. Create Account for Manager & Customer

- *Mobile*

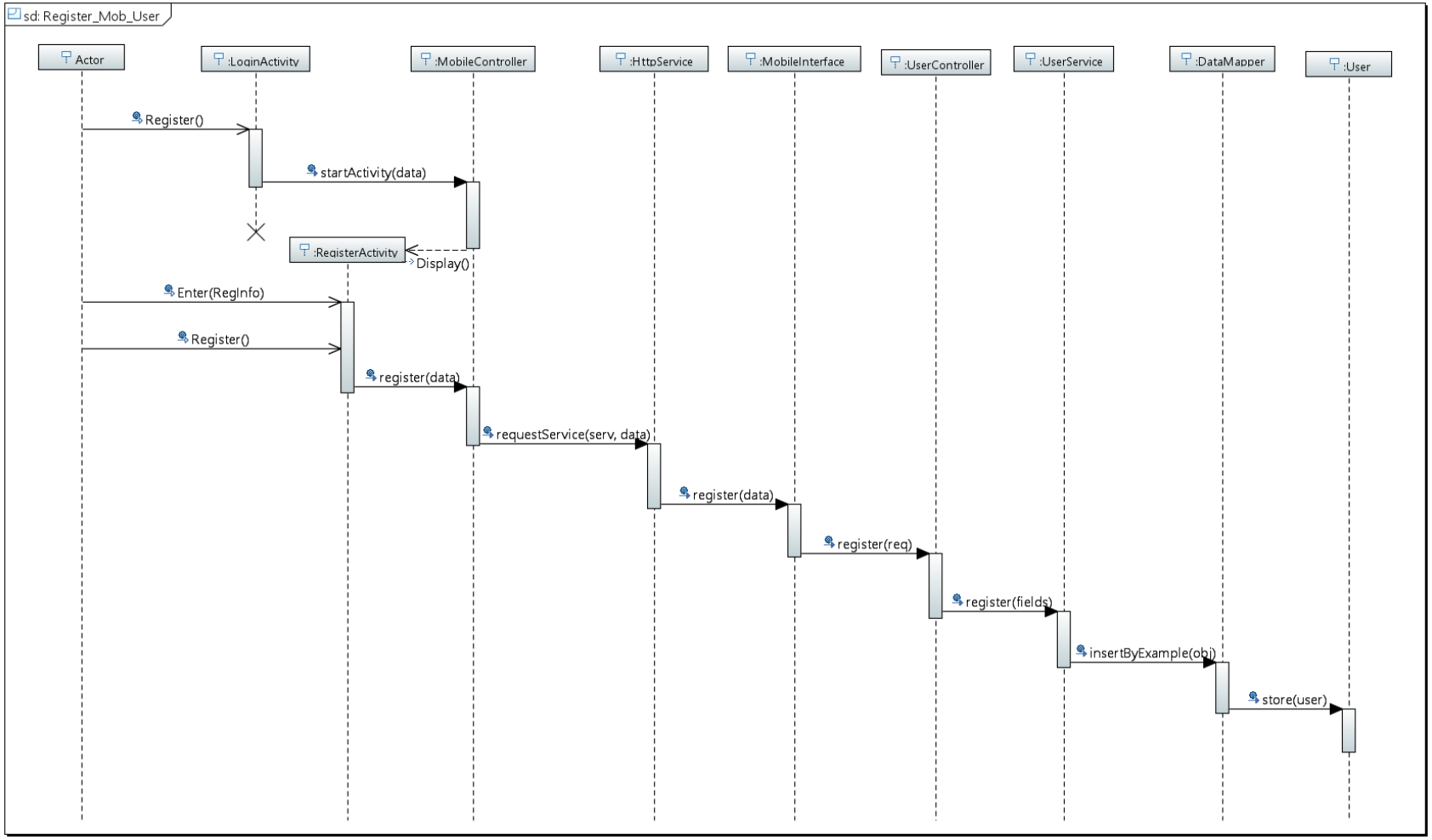


Figure 6.11. Sequence diagram of "Create Account for Manager & Customer" use case (Mobile Interface).

In Figure 6.11 above, the mobile deployment of this registration page is used primarily by customers. The customer requests the registration page, and enters the pertinent information. The data transmits through the client tier into the logic tier through the UserController. The data is then mapped into persistent data and ultimately reaches the data store tier as a User entity.

- *Web*

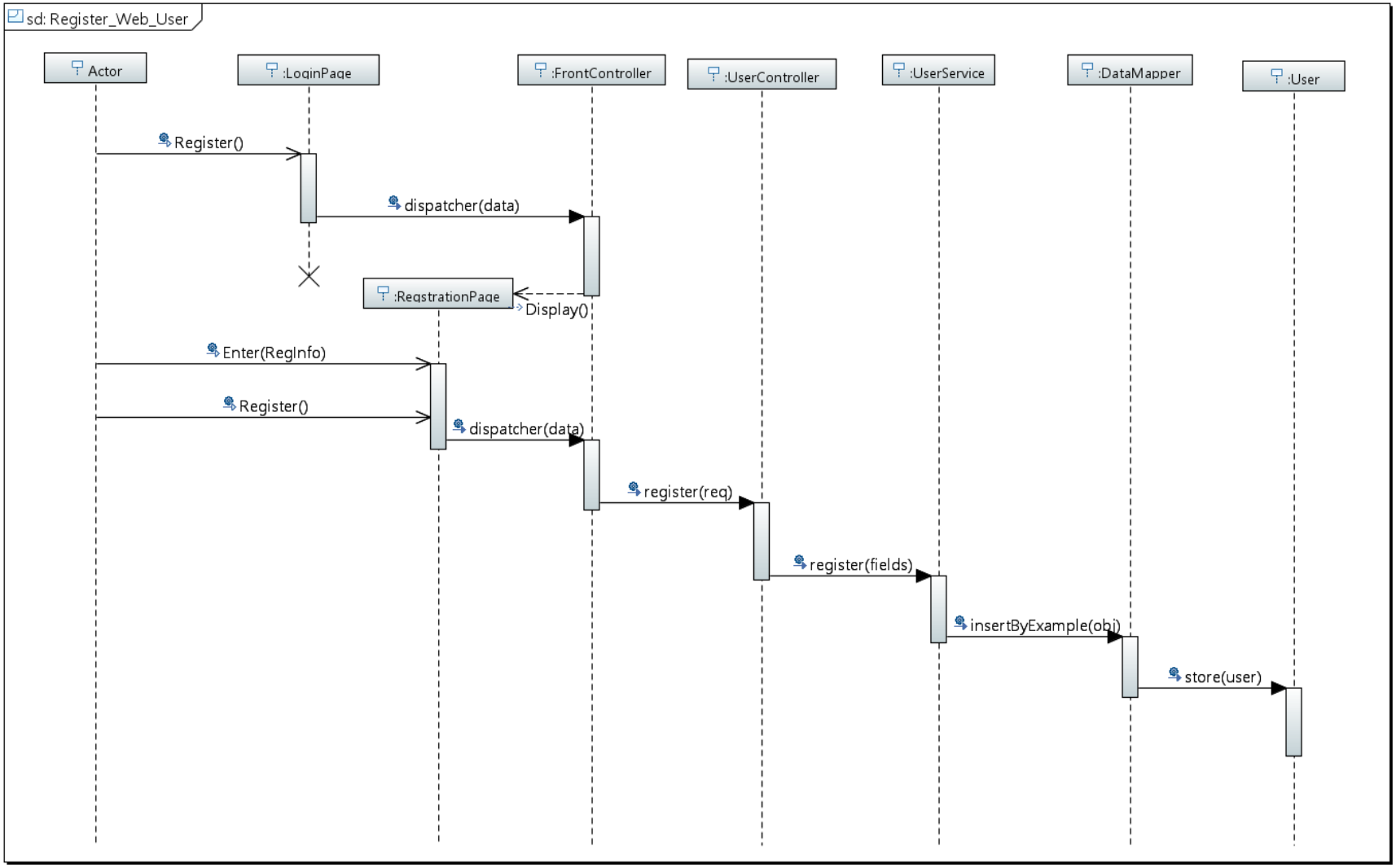


Figure 6.12. Sequence diagram of "Create Account for Manager & Customer" use case (Web Interface).

The web deployment shown in Figure 6.12 works in a way similar to the mobile deployment shown above. The user requests the registration page, and enters information. The data then progresses through the various tiers in the system and ultimately is incorporated as a User entity in persistent data storage.

2. Login System for User

- *Web*

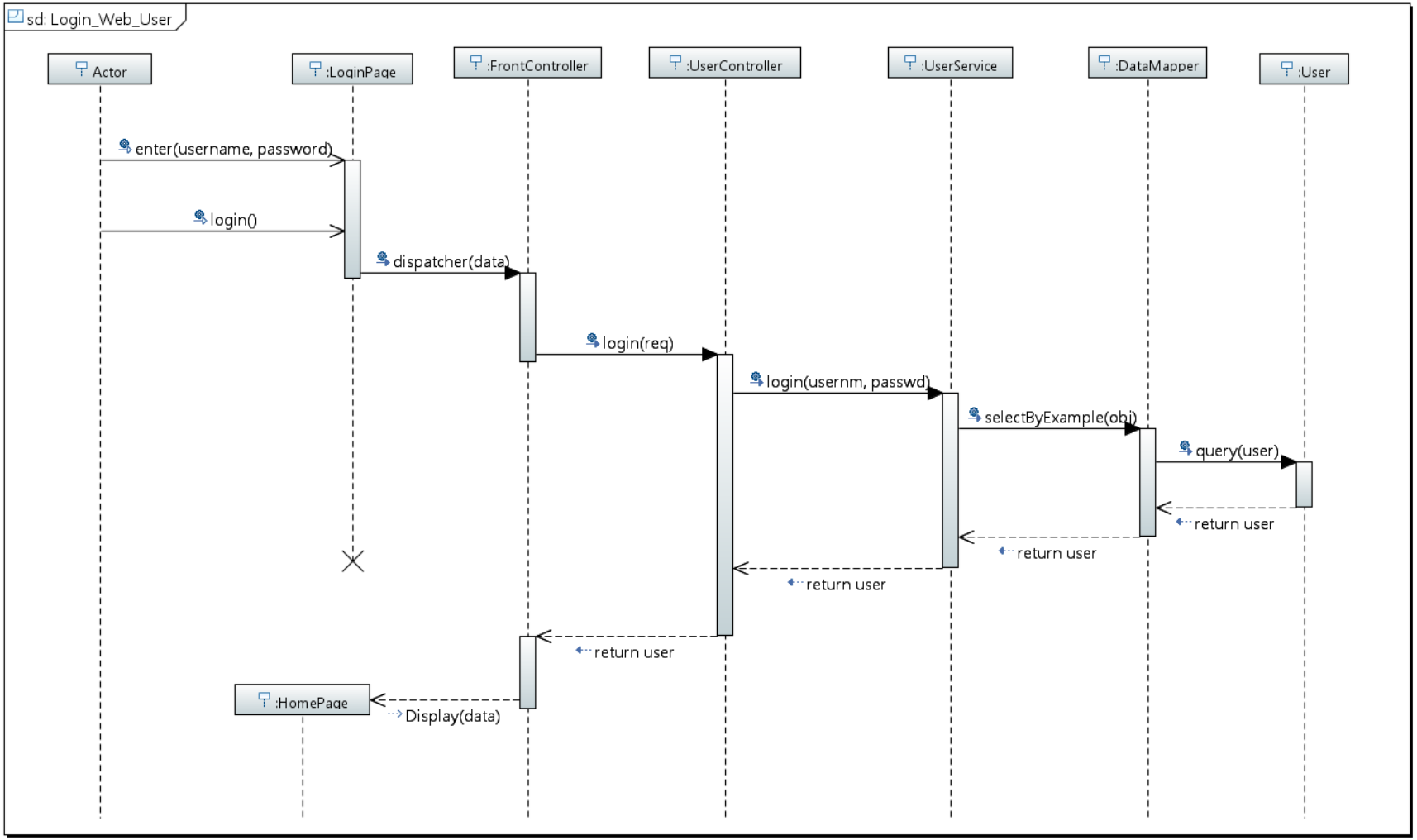


Figure 6.13. Sequence diagram of "Login System for User" use case (Web Interface).

In Figure 6.13 above, an actor logs into the system by entering the corresponding username and password and presses the button "Login". After this, the FrontController receives the request and sends it to the UserController that calls the function "login" with the information entered by the user in UserService. It then calls the function selectByExample of the data access interface DataMapper to look up the user in the database. Finally, once the information is validated, the FrontController displays the "HomePage".

- *Mobile*

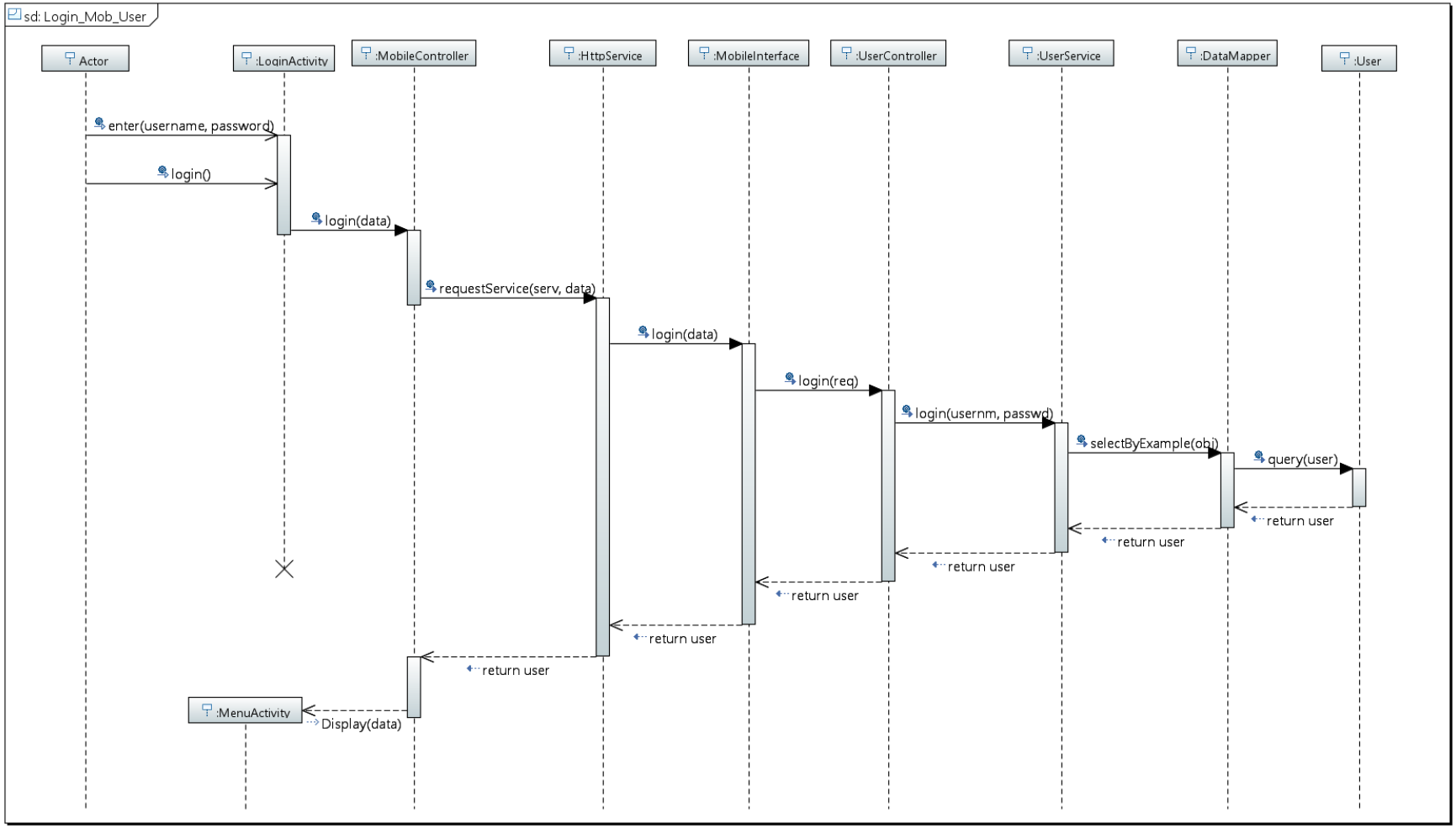


Figure 6.14. Sequence diagram of "Login System for User" use case (Mobile Interface).

In Figure 6.14 above, an actor logs into the system by entering the corresponding username and password and pressing the button "Login". After this, the MobileController receives the request and sends it to the HttpService which sends the request to the UserController through the MobileInterface. The UserController calls the function "login" with the information entered by the user in UserService, then calls the function selectByExample of the data access interface DataMapper to look up the user in the database. Finally, once the information is validated, the MobileController display the "MenuActivity" with the functionality.

3. Logout System for User

- *Web*

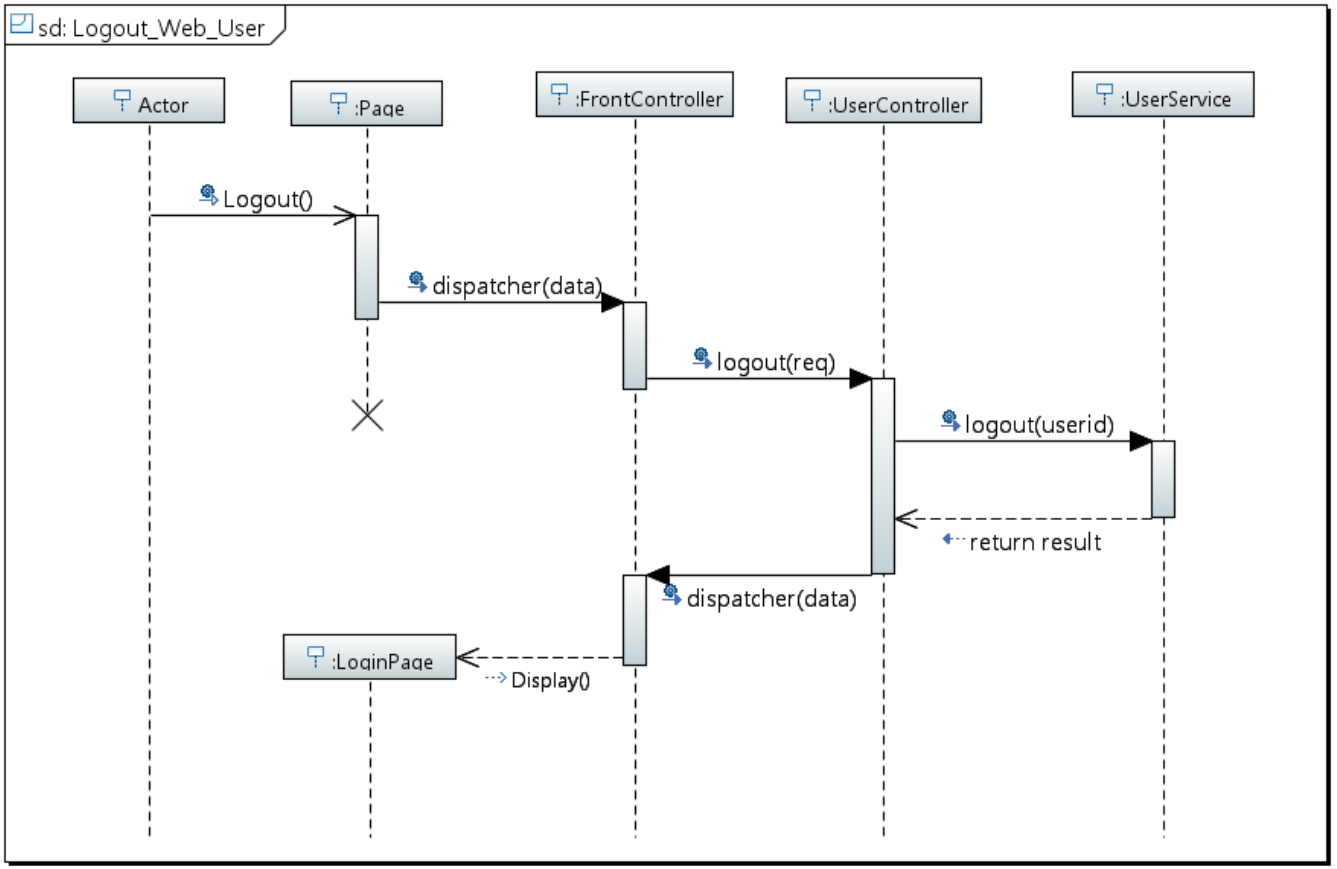


Figure 6.15. Sequence diagram of "Logout System for User" use case (Web Interface).

In Figure 6.15 above, an actor clicks the button "logout" to be logged out of the system. The FrontController receives the request and sends it to the UserController that calls the function "logout" in UserService, then the FrontController displays the "LoginPage".

- *Mobile*

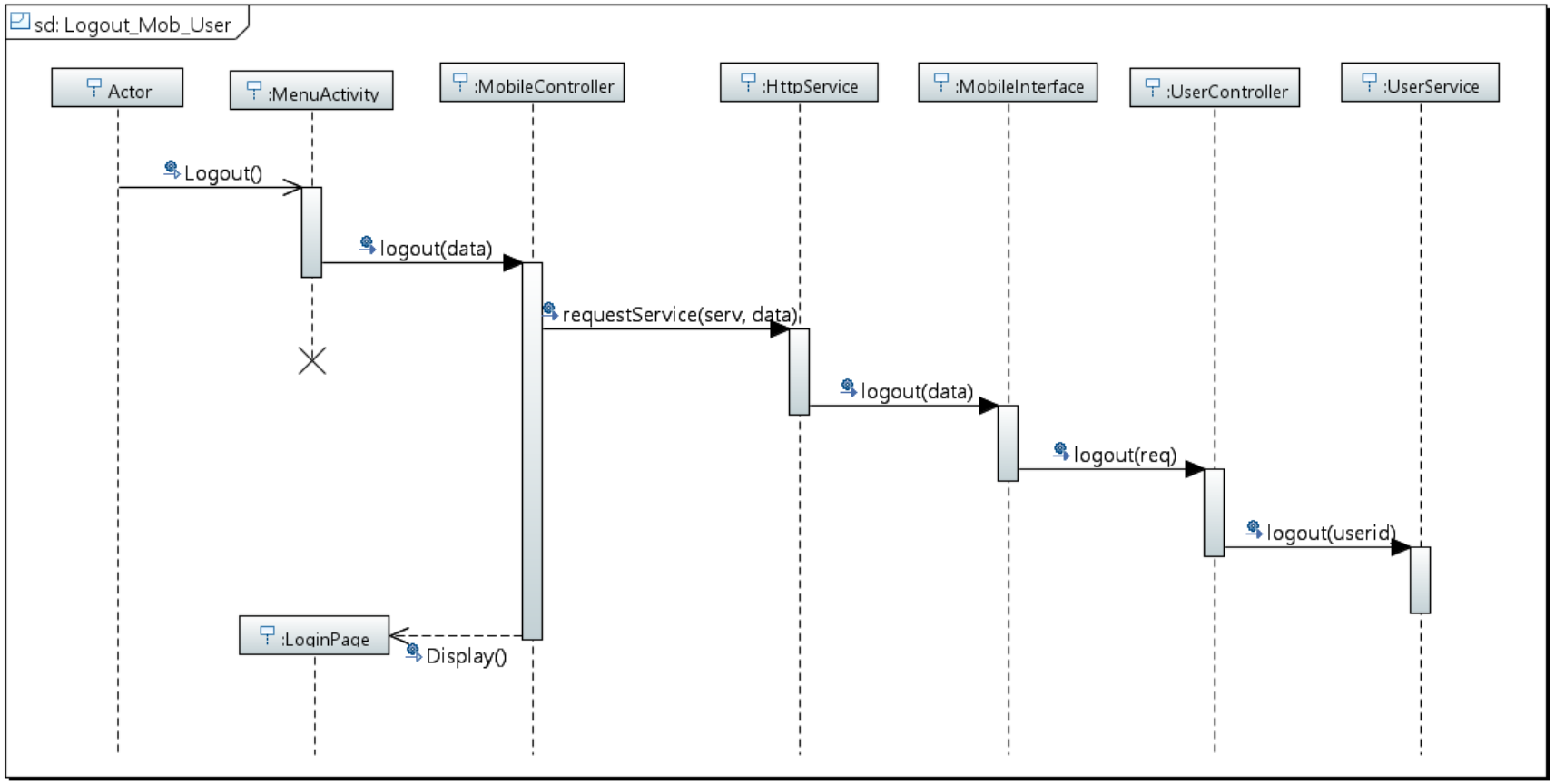


Figure 6.16. Sequence diagram of "Logout System for User" use case (Mobile Interface).

In Figure 6.16 above, an actor clicks the button "logout" to be logged out of the system. After this, the MobileController receives the request and sends it to the HttpService that sends the request to the UserController through the MobileInterface. The UserController calls the function "logout" in UserService, then the MobileController displays the "LoginPage".

4. Roles Control for User

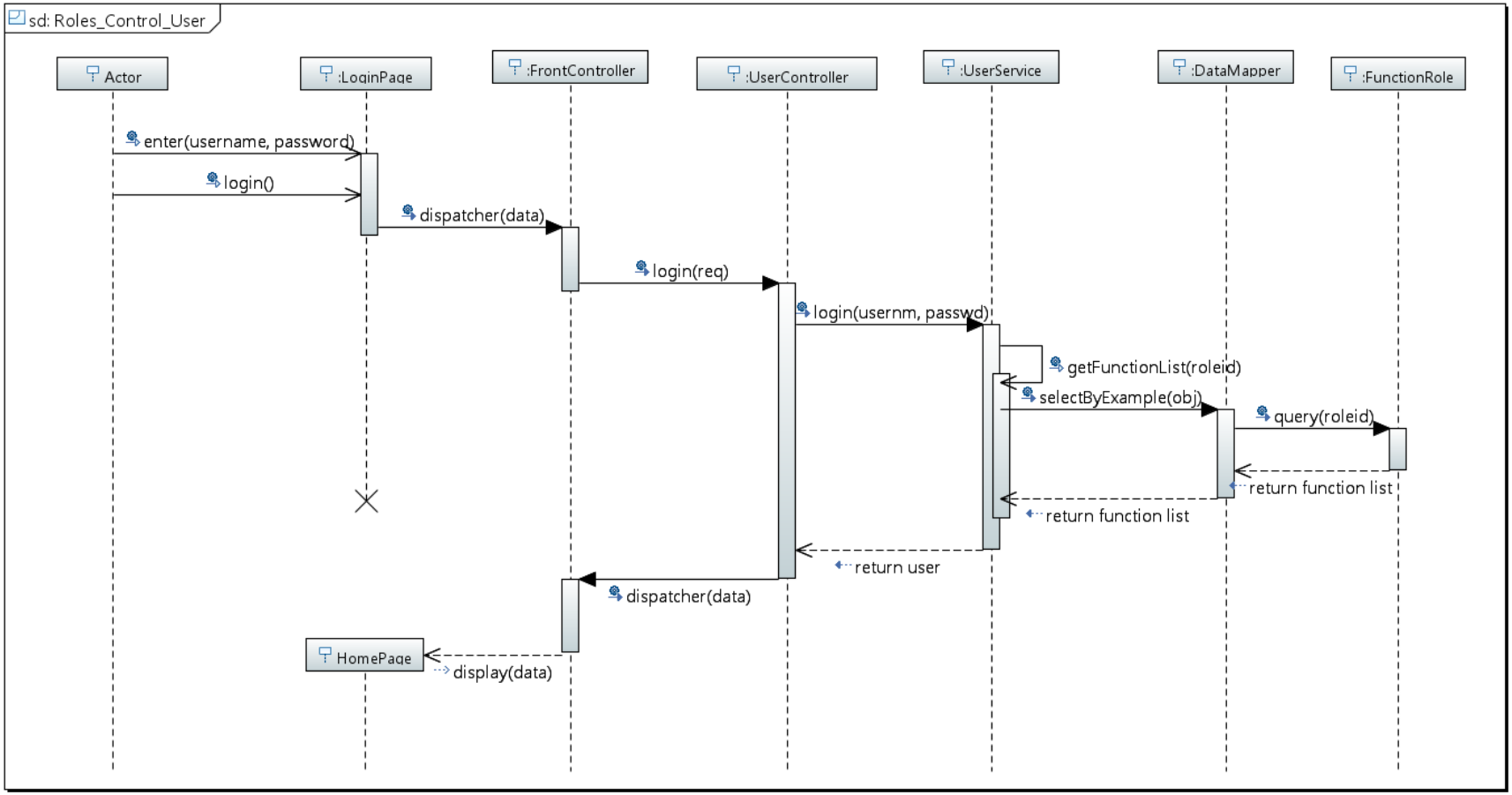


Figure 6.17. Sequence diagram of "Roles Control for User" use case.

In Figure 6.17 above, an actor enters username and password, then clicks the "login" button on Login page to trigger function login. The FrontController receives the request and transmits it to UserController that calls function login with username and password in UserService. UserService call function getFunctionList with roleid, then call function selectByExample of a data access interface DataMapper to get the functions which are accessible to the current actor. Finally, the user's information is returned and displayed on the HomePage.

5. Visualize Products for Staff

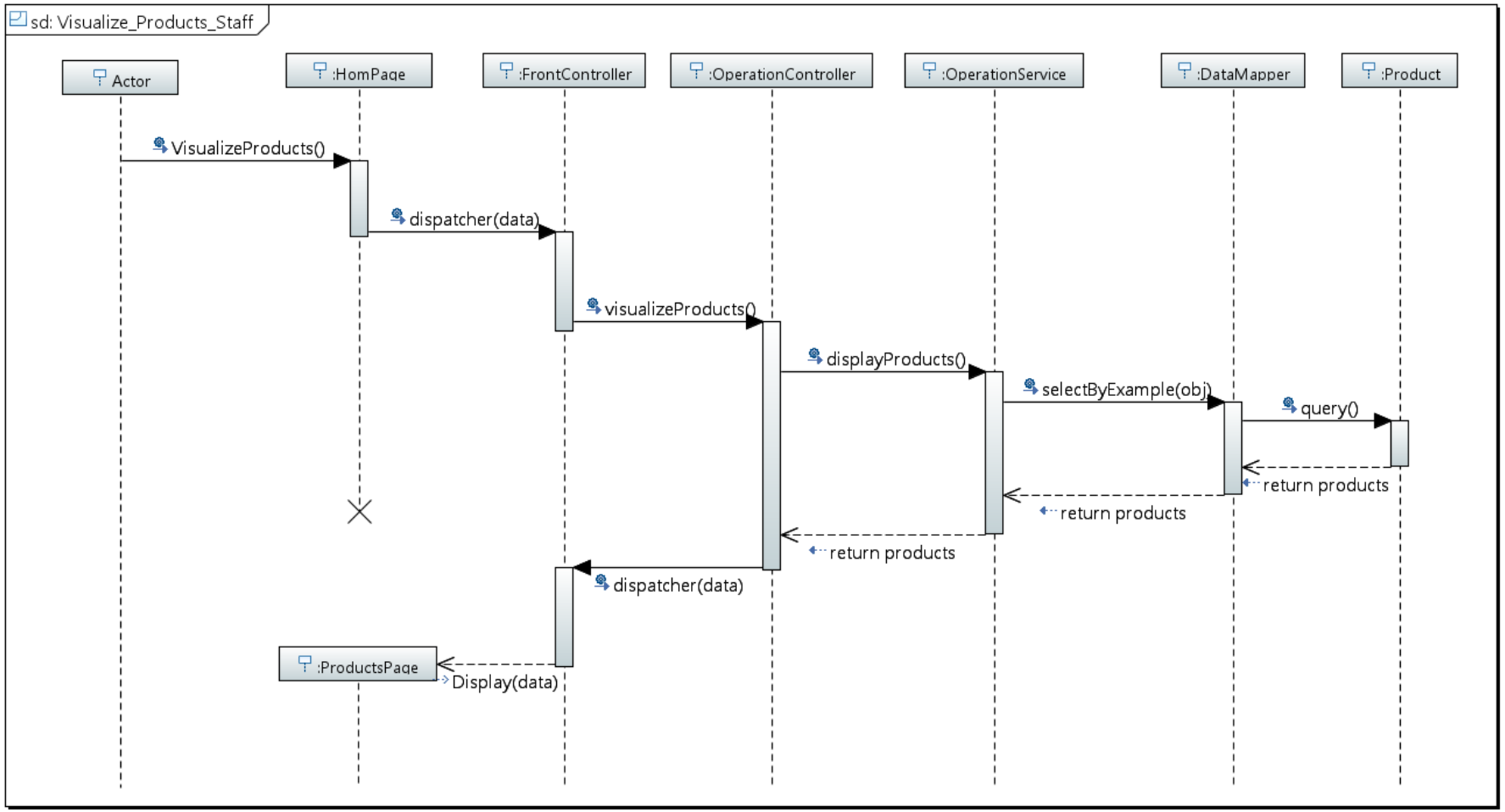


Figure 6.18. Sequence diagram of "Visualize Products for Staff" use case.

In Figure 6.18 above, an actor selects the option "VisualizeProducts" to see a list of products. After this, the FrontController receives the request and sent it to the OperationController that calls the function "displayProduct" in OperationService, then call the function selectByExample of a data access interface DataMapper to look for the list of products into the database. Finally, the Product return the product list that is displayed by the FrontController in the "ProductPage".

6. Add Product for Staff

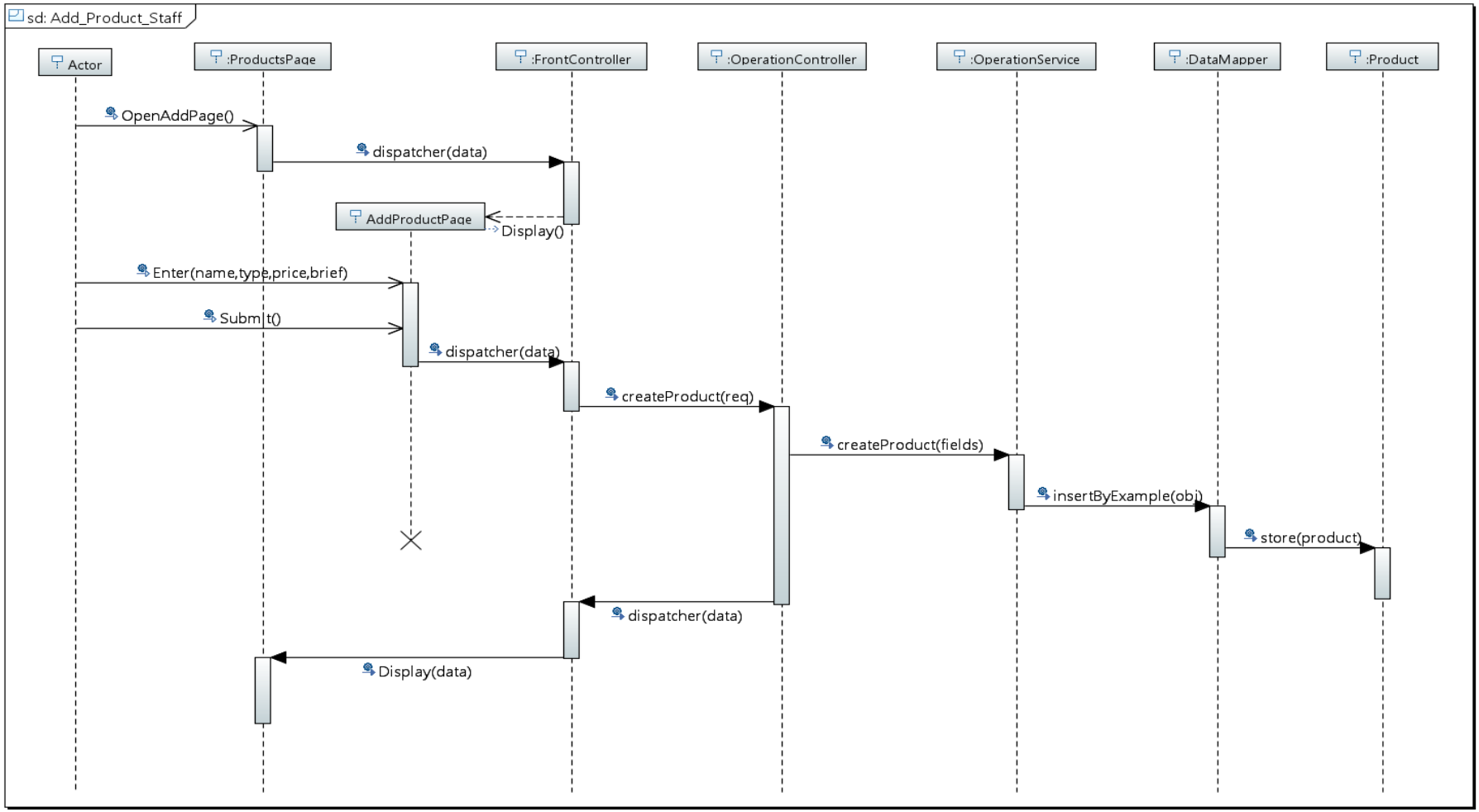


Figure 6.19. Sequence diagram of "Add Product for Staff" use case.

In Figure 6.19 above, an actor clicks the "Add" button on Products page to trigger the function OpenAddPage. This calls the function dispatcher with the address of the AddProduct page in the FrontController. After that, the AddProduct page is displayed. The actor enters information of a new product and submits it. The FrontController receives the request and transmits it to OperationController that calls the function createProduct with field values of the new product in OperationService. It then calls the function insertByExample of the data access interface DataMapper to insert the new product into the database. Finally, the new product shows on a product list.

7. Visualize Menu for Customer

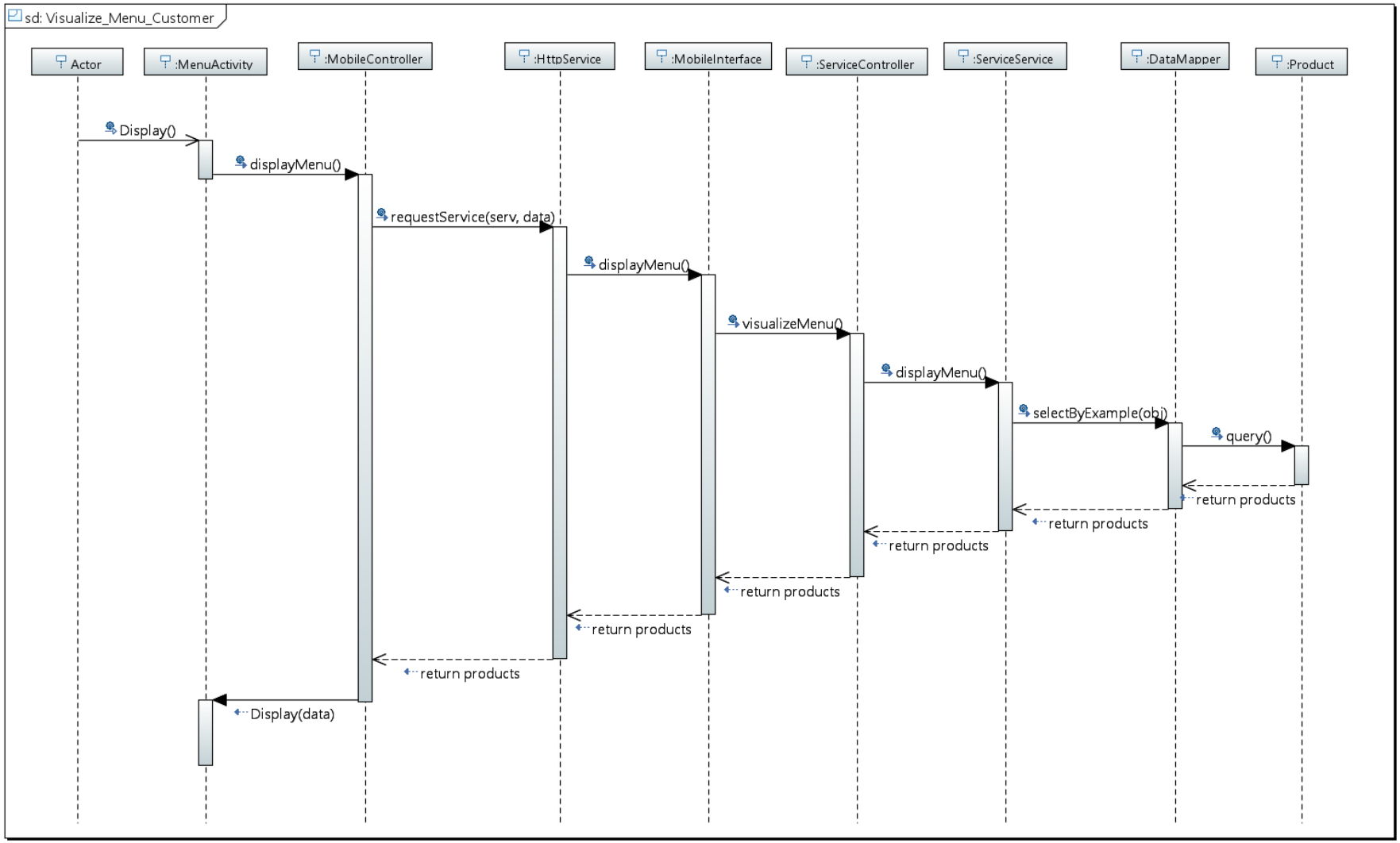


Figure 6.20. Sequence diagram of "Visualize Menu for Customer" use case.

In Figure 6.20 above, an actor selects "VisualizeMenu" to visualize the menu. After this, the MobileController receives the request and sends it to the ServiceController through the MobileInterface. The UserController calls the function "displayMenu" in ServiceService, then calls the function selectByExample of the data access interface DataMapper to look up the list of products of the menu from the database. Finally, Product returns the product list that is displayed by the MobileController in the "MenuActivity".

8. Add Product for Customer

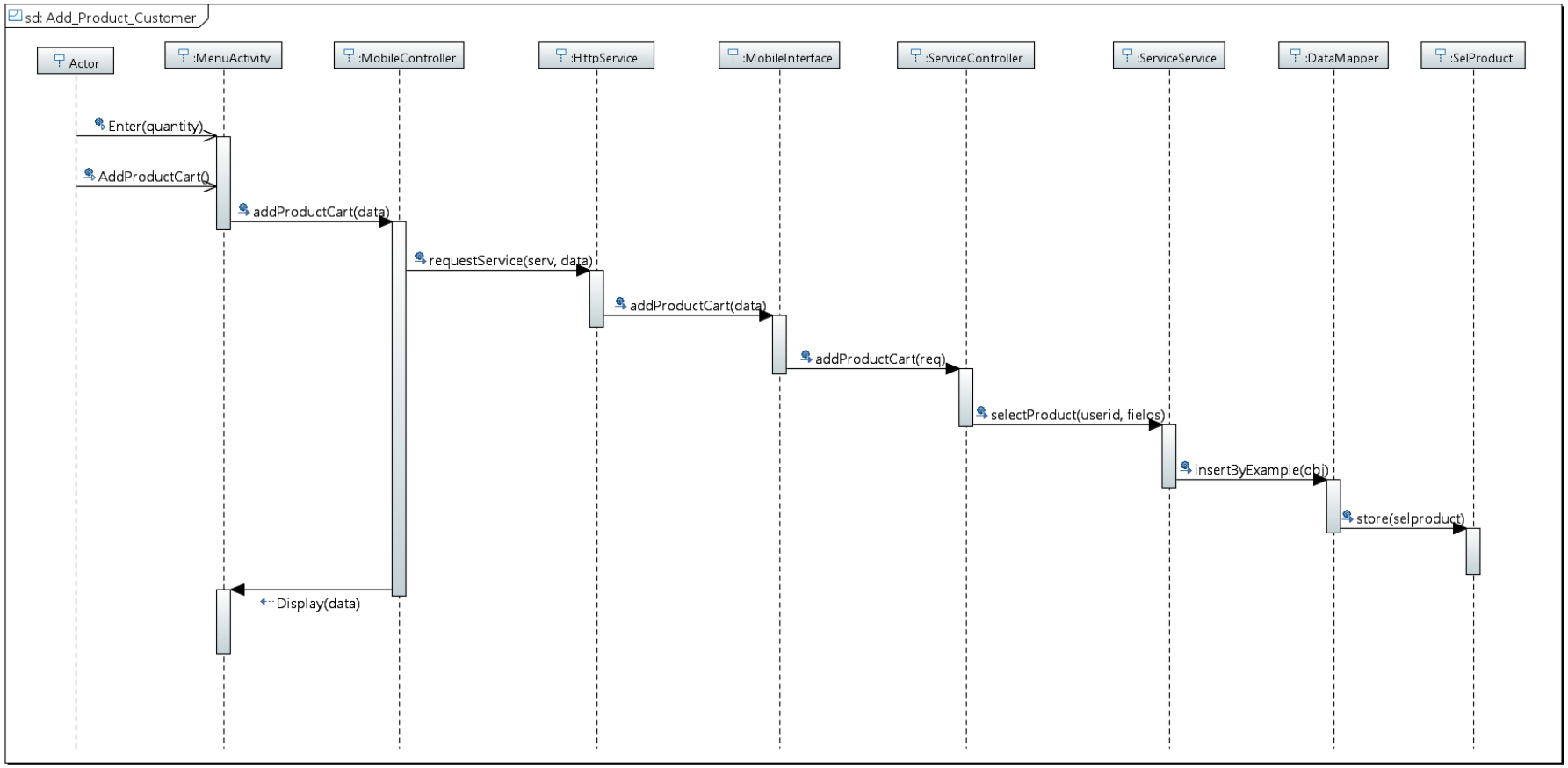


Figure 6.21. Sequence diagram of "Add Product for Customer" use case.

In Figure 6.21 above, an actor enters the quantity of a product he wants to purchase and clicks the "Add" button on Menu page. This executes the function AddProductCart that calls the function dispatcher with the selected product data in MobileController. The MobileController sends the request to MobileInterface through HTTP, then transmits it to the FrontController. The FrontController receives the request and transmits it to the ServiceController that calls function selectProduct with field values of the product in ServiceService, then calls function insertByExample of the data access interface DataMapper to insert the selected product into the database. Finally, number of products in the actor's shopping cart is updated.

9. View Cart for Customer

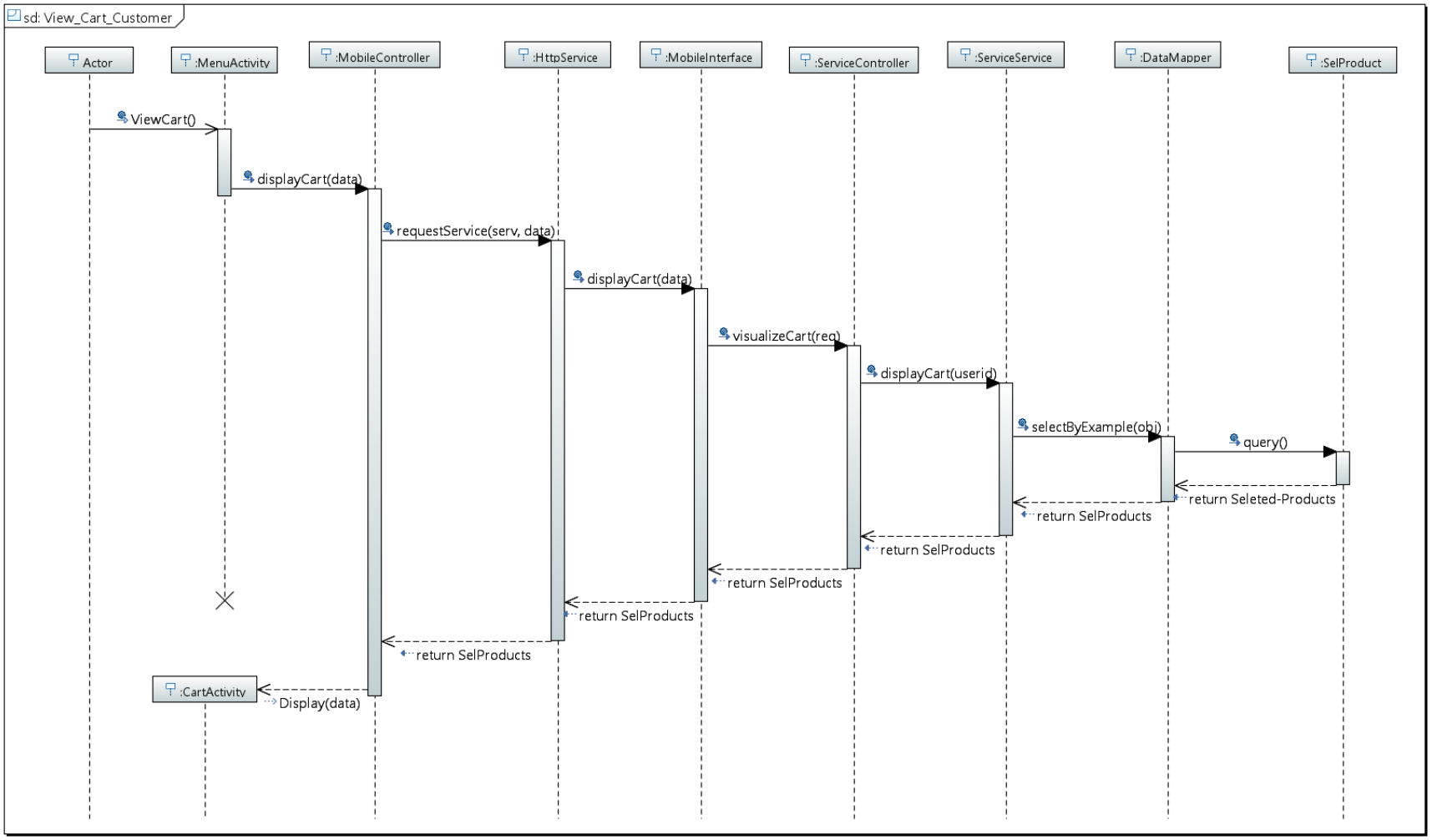


Figure 6.22. Sequence diagram of "Roles Control for User" use case.

In Figure 6.22 above, an actor clicks the "My cart" button on menu to trigger function ViewCart, then it calls function dispatcher with the user data in MobileController. The MobileController sends the request to MobileInterface through HTTP, then transmits to ServiceController. The ServiceController receives the request and calls function displayCart with userid in ServiceService, then calls function selectByExample of the data access interface DataMapper to search for the selected products in the database. Finally, the selected products are returned and displayed in the Cart.

10. Remove Product for Customer

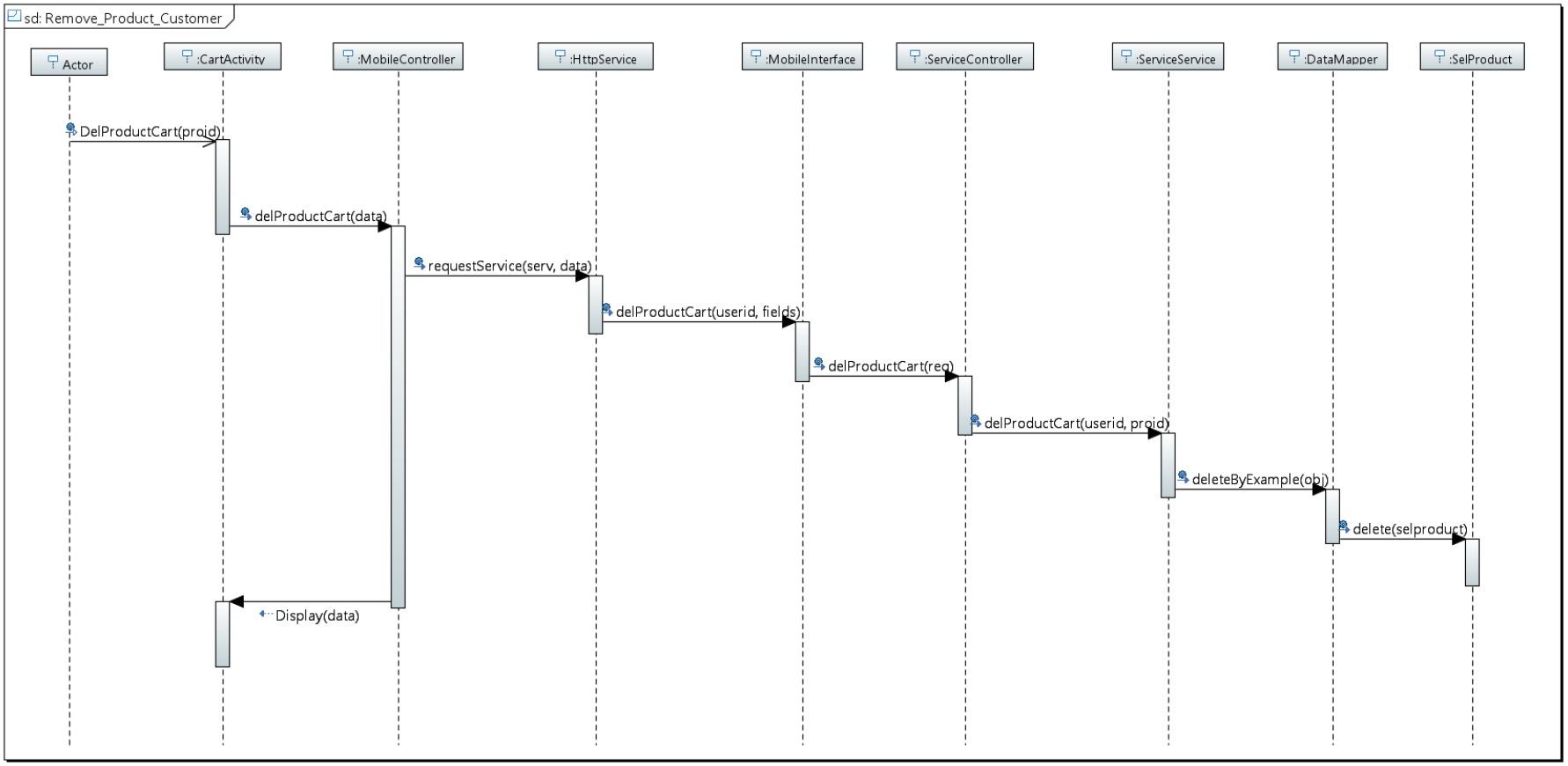


Figure 6.23. Sequence diagram of "Remove Product for Customer" use case.

Figure 6.23 shows the sequence diagram for the "Remove Product" use case. The request information flows from the client layer in the MobileController into the ServiceController, which relays information through the Logic level in the ServiceService object and finally the entity is deleted at the database level in the selProduct entity.

11. Fill Out Order Information for Customer

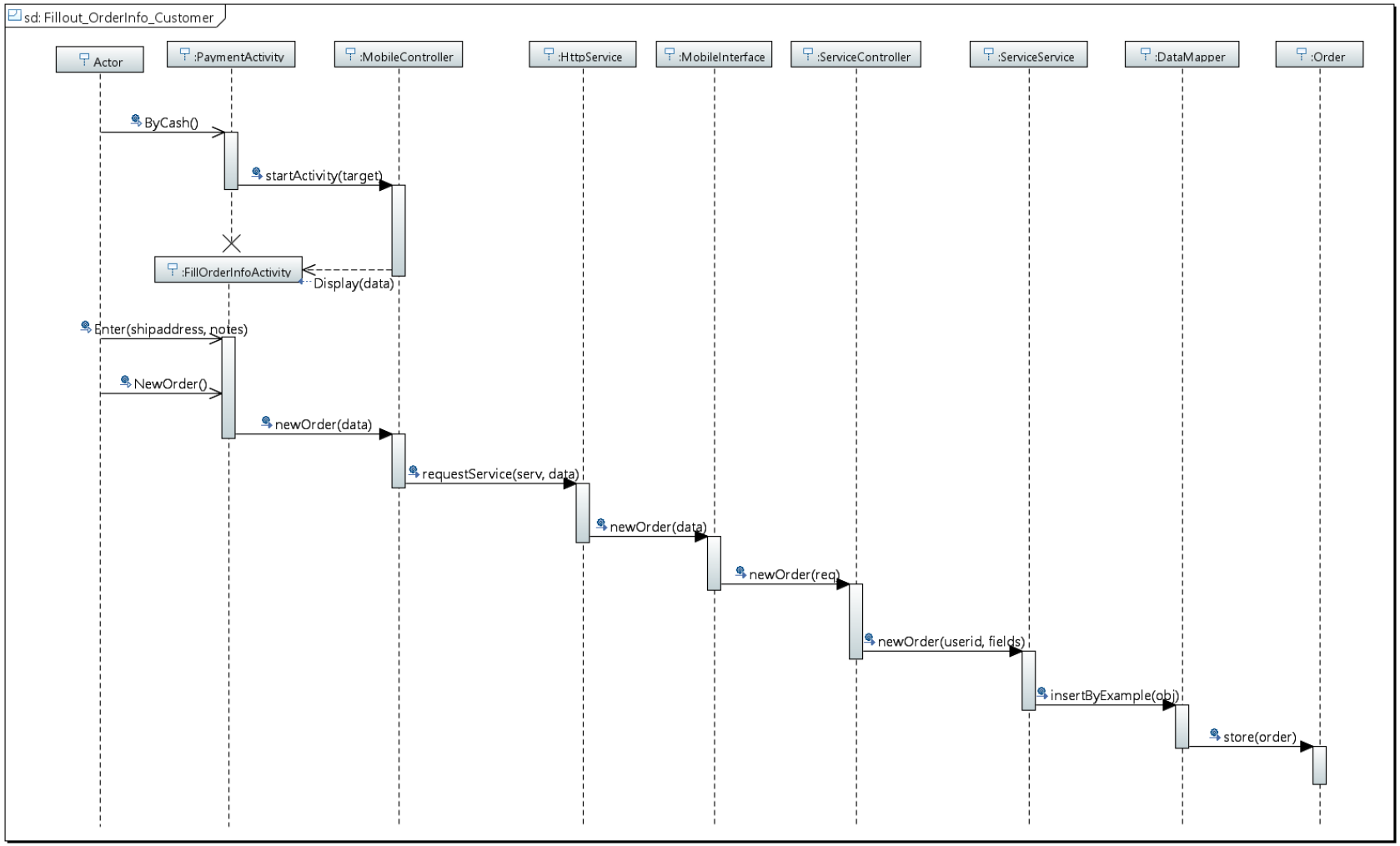


Figure 6.24. Sequence diagram of "Fill out Order Information for Customer" use case.

In Figure 6.24 above, an actor presses the "cash" button, then the MobileController receives the request and responds by displaying the fields that the actor needs to fill. The actor enters shipping address and delivery notes and presses the "place order" button. The MobileController receives the request and sends it to the HttpService which in turn forwards it to the ServiceController through the MobileInterface. The ServiceController calls the function "newOrder" with the actor's user id and the fields entered by the actor. ServiceService calls the function insertByExample to create the order and the DataMapper calls the function "store" passing the order.

12. Pop Address Alert for Customer

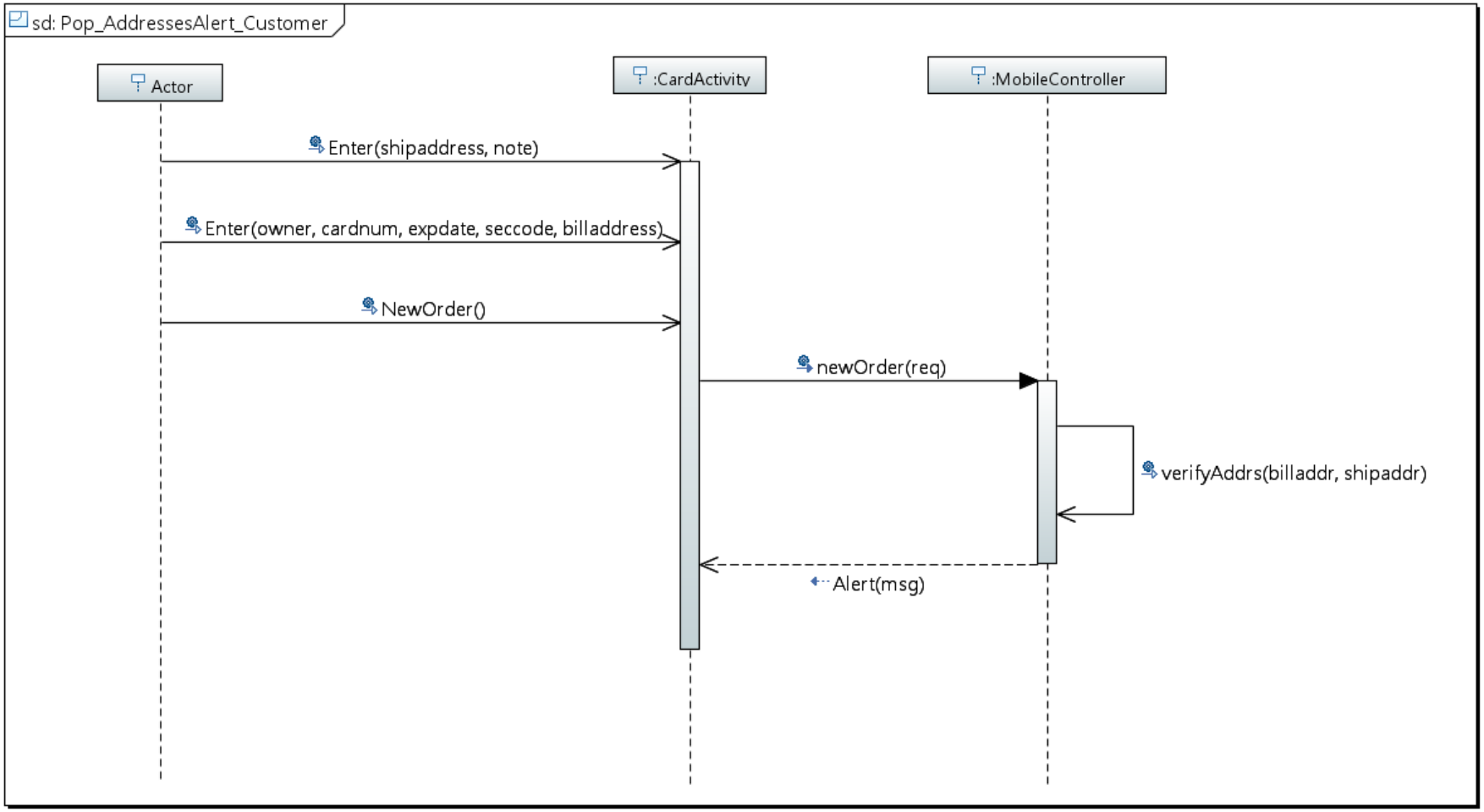


Figure 6.25. Sequence diagram of "Pop Address Alert for Customer" use case.

In Figure 6.25 above, the actor enters the shipping address and delivery notes. The actor also enters credit card information such as owner's name, card number, expiration date, security code, and billing address. Actor then presses the "submit order" button to place the order. The request is received by "CardAcivity" which forwards it to the MobileController. The MobileController calls the function "verifyAddrs" passing the actor's billing and shipping address as parameters and verifying consistency. Because the two addresses entered by the actor differ, the "verifyAddrs" function detects the inconsistency and sends an alert message to CardActivity by calling the "Alert" function.

6.4. Detailed Class Design

In this section, we talk about the detailed class design of our system. We first describe the purpose of the classes in each subsystem, including Client subsystem, Presentation subsystem, Application logic subsystem, and Data Store subsystem. Then we introduce the OCL constraints for the class invariants in each subsystem.

6.4.1. Purpose of Classes

1. Native Mobile Application Class Diagram

The Native Mobile Application class diagram in Figure 11.4.1 represents the MVC architecture of the mobile client sub-system. The HttpService Model directly manages the data, logic, and rules. It stores the state of the subsystem and provides ways to query and change such state. The MobileController Controller handles authentication, session management, and incoming user requests by translating them into actions that the HttpService Model should take. Such actions include registration, log in, log out, displaying the menu, displaying, adding and removing products from the shopping cart, verifying address consistency, and placing orders.

The Activity View represents the User Interface. At its most general level, it displays data, alerts, and requests the visualization of the shopping cart. The Controller is able to select different specifications of the View. SplashActivity displays the title and version of the subsystem. LoginActivity visualizes fields for username and password and provides login and registration functionality. RegisterActivity provides fields for username, password, first and last name, phone number, security question and answer, and user role.

MenuActivity lists the menu and item quantity, enabling the user to enter the quantity of a desired product, adding it to the shopping cart. CartActivity allows the user to visualize the list of chosen products and their quantity, as well as to remove unwanted product. PaymentActivity allows the user to select the payment method, that is, cash or credit/debit card. FillOrderInfoActivity allows the user to enter phone number, shipping address, and notes for delivery. It provides the functionality to place the order if the user chose to pay with cash. CardActivity allows the user to enter additional information such as card name, card number, expiration date, security code, billing address. It allows the user to place the order with

credit/debit card and it displays an alert if there is an inconsistency between shipping and billing address.

2. Presentation Class Diagram

The Presentation class diagram in Figure 11.4.2 is part of the 4-Tier primary architecture pattern of the HMTS system. It encapsulates the presentation logic utilized in servicing users by intercepting requests, conducting section management, controlling access to business services, and constructing/delivering responses. The Page Boundary produces the UI elements for user interaction based on the user id (in turn linked to the user role). It includes functionality to load and display data, and to log out of the HMTS system. It generalizes a variety of pages.

HomePage provides functionality to visualize products, shopping cart, and placed orders. It in turn extends MenuForm, intended to display menu items. LoginPage provides functionality for the user to enter a username and password and to register, login, or reset a forgotten password. RegistrationPage provides functionality for storing a user's username, password, first and last name, phone number, security question and answer, and role. ProductsPage provides functionality to display a list of products, as well as to request the addition/removal of products. AddProductPage allows the addition of a new product unto the menu, to include product name, type, price, and a brief description. MobileInterface provides the logic for the mobile subsystem, including functionality such as registration, login/logout, displaying menu, displaying, adding, and deleting products from shopping cart, and placing a new order. Data allows the FrontController to interact with data.

3. ApplicationLogic Class Diagram

The ApplicationLogic class diagram in Figure 11.4.3 represents the business logic required by the HMTS system. Functionality such a displaying the menu, selecting a product, or registering a new user has behind the scenes a variety of transactions which are represented in this class diagram. The main focus is on building users, operations, and services, as well as providing registration capability for web and mobile, differentiating between what each user can visualize and what actions can each user perform based on user role, and specify data transfer and service operation.

4. DataStore Class Diagram

The DataStore class diagram in Figure 11.4.4 represents the subsystem responsible for communication with external resources and systems. DataMapper provides data interaction functionality to select by example, primary key, insert by example, update by example, and delete by primary key. The SqlSessionFactory interface allows to open a SQL section, extending SqlSessionManager, SqlSessionTemplate, and the SqlSession interface. These allow the creation of a new session instance, the opening/closing of a new session, and the execution of insertion, selection, deletion, and update of data. A number of entities are represented in this diagram. User represents a customer, and it includes fields such as id, username, password, first and last name, phone number, and role. It extends SecQuestion providing the capability to verify a security question as a second level of security in authentication or password reset.

The FunctionRole entity defines the functions that a given user can access based on his user role. The SelProduct entity specifies the quantity of a given product, and is in a many-to-one relationship with the Product entity, including product id, name, type, price, and brief description. The latter is a product item on the menu, not to be confused with OrderProduct, representing a product in a specific order processed for a given user. Order represents an order placed by a customer, including id, phone number, shipping address, delivery notes, and status. OrderProduct is in a many-to-one relationship with Order, and includes id, name, type, price, and brief description of a given product ordered by a customer.

Payment includes functionality about payment session id, method (cash/card), and amount. Card specifies additional capability for credit/debit card processing, such as card unique id, card number and name, expiration date, security code, billing address, and consistency of the latter with a user's shipping address for increased security.

For the class interfaces of the subsystems, see Appendix E.

6.4.2. Object Constraints Language

This section highlights the OCL constraints on the various subsystem control objects in the system. The subsystems introduced are Client, Presentation, ApplicationLogic, and DataStore. See OCL of class diagrams in Appendix D.

- **Client Subsystem – MobileController**

This controller exists at the mobile client and is responsible for login, and manipulating customer inputs into objects to be sent to the application logic tier.

context MobileController::login(data:Json)
inv: data->includes (username) and data->includes(password)
pre: Activity::UserID = null
post: if SUCCESS then
    Activity::UserID <> null
else
    Activity::UserID = null
endif

context MobileController::addProduct(data:Json)
inv: data->includes(userid) and data->includes(product)
pre: Not CartActivity::Products->includes(product)
post: CartActivity::Products->includes(product)

context MobileController::newOrder (data:Json)
inv: data->includes(userid) and data->includes(products)
pre: products->forAll(p:products | CartActivity::Products->includes(p))
post: products->isEmpty()

- **Presentation Subsystem – FrontController**

This controller exists at the presentation layer and is responsible for forwarding mobile requests/responses and building pages for web clients.

context FrontController::dispatcher(request:Data)
pre: request <> null
post: result <> null and result->includes(url)

- **Application Logic Subsystem – ServiceService**

This controller exists within the application logic and is responsible for building the objects

that are sent to the database layer for storage, the objects sent to the presentation layer for client interface, and the internal logic of the system.

context ServiceService::selectProduct (userid:String,productfields:String[])
inv: userid <> null and fields <> null
pre: ShoppingCart <> null
post: product = productfields and ShoppingCart->includes(product)

context ServiceService::delProduct (userid:int,productid:int)
inv: userid <> null and productid <> null
pre: ShoppingCart::Product::ProductID->includes(productid)
post: Not ShoppingCart::Product::ProductID->includes(productid)

context ServiceService::newOrder (userid:int,productsfields:String[])
inv: userid <> null and productsfields <> null
pre: productsfields->forAll(p: productsfields | ShoppingCart->includes(p))
post: productsfields->forAll(p: productsfields | OrderProduct->includes (p))

- **DataStore Subsystem – Data Mapper**

This controller exists at the database tier and is responsible for maintaining data and sending/retrieving information from application logic.

context DataMapper::selectByExample(obj:DataExample):List
inv: obj <> null
post: result->size() >= 0

context DataMapper::insertByExample(obj:DataExample)
inv: data.oclAsType(obj)
pre: obj <> null

7. Testing Process

In this chapter we introduce the testing process of our system. Our testing process includes unit tests, subsystem tests and system tests. After that, we evaluate the tests by giving all the test results, and the code coverage for the testing.

7.1. Unit Tests

In this section, we describe the unit tests of our system. The test cases are shown in the tables below.

Test Case ID	UNIT01_HMTS_CreateAccount01
Purpose	Testing operation of UserService.register function for mobile service.
Test Setup	UserService is initialized.
Input	Pass a User object as a parameter: • User.firstName: “Bob” • User.lastName: “Hope” • User.username: “bobhope@gmail.com” • User.password: “bobhope13” • User.phone: “(213)112-2432” • User.role: 4
Expected Output	Return a User object: • User.firstName: “Bob” • User.lastName: “Hope” • User.username: “bobhope@gmail.com” • User.password: “bobhope13” • User.phone: “(213)112-2432” • User.role: 4

Test Case ID	UNIT01_HMTS_CreateAccount02
Purpose	Testing operation of UserService.register function without username value for mobile service.
Test Setup	UserService is initialized.
Input	Pass a User object as a parameter: • User.firstName: “Bob” • User.lastName: “Hope” • User.username: “” • User.password: “bobhope13” • User.phone: “(213)112-2432” • User.role: 4
Expected Output	Return a User object: • User.firstName: NULL • User.lastName: NULL • User.username: NULL • User.password: NULL • User.phone: NULL • User.role: NULL

Test Case ID	UNIT01_HMTS_CreateAccount03
Purpose	Testing operation of UserService.register function for web service.
Test Setup	UserService is initialized.
Input	Pass a User object as a parameter: • User.firstName: “Peter” • User.lastName: “John” • User.username: “peter@gmail.com” • User.password: “123456” • User.phone: “(305)837-4827” • User.role: 2
Expected Output	Return a User object: • User.firstName: “Peter” • User.lastName: “John” • User.username: “peter@gmail.com” • User.password: “123456” • User.phone: “(305)837-4827” • User.role: 2

Test Case ID	UNIT01_HMTS_CreateAccount04
Purpose	Testing operation of UserService.register function without username value for web service.
Test Setup	UserService is initialized.
Input	Pass a User object as a parameter: • User.firstName: “Peter” • User.lastName: “John” • User.username: “” • User.password: “123456” • User.phone: “(305)837-4827” • User.role: 2
Expected Output	Return a User object: • User.firstName: *NULL* • User.lastName: *NULL* • User.username: *NULL* • User.password: *NULL* • User.phone: *NULL* • User.role: *NULL*

Test Case ID	UNIT01_HMTS_CreateAccount04
Purpose	Testing operation of UserService.register function with invalid user type value.
Test Setup	UserService is initialized.
Input	Pass a User object as a parameter: • User.firstName: “Peter” • User.lastName: “John” • User.username: “peter@gmail.com” • User.password: “123456” • User.phone: “(305)837-4827” • User.role: -1
Expected Output	Return a User object: • User.firstName: NULL • User.lastName: NULL • User.username: NULL • User.password: NULL • User.phone: NULL • User.role: NULL

Test Case ID	UNIT02_HMTS_Login01
Purpose	Verify method UserService.Login can login into the system successfully for web service.
Test Setup	UserService and User object is initialized.
Input	Object user (an instance of class User): • user.username: “mary@fiu.edu” • user.password: “my2882” • user.type: “2” • user.tag: “1,2,”
Expected Output	Return a User object: • user.username: “mary@fiu.edu” • user.password: “my2882” • user.type: “2” • user.tag: “1,2,”

Test Case ID	UNIT02_HMTS_Login02
Purpose	Verify method UserService.Login can deal with exceptions when enter invalid data for web service.
Test Setup	UserService and User object is initialized.
Input	Object user (an instance of class User): • user.username: “nick” • user.password: “my2882” • user.type: “2”
Expected Output	Return a User object: • user.userId: -1

Test Case ID	UNIT02_HMTS_Login03
Purpose	Verify method UserService.Login can login into the system successfully for mobile service.
Test Setup	UserService and User object is initialized.
Input	Object user (an instance of class User): • user.username: “david@yahoo.com” • user.password: “123456” • user.type: “4”
Expected Output	Return a User object: • user.username: “david@yahoo.com” • user.password: “123456” • user.type: “4”

Test Case ID	UNIT02_HMTS_Login04
Purpose	Verify method UserService.Login can deal with exceptions when enter invalid data for web service.
Test Setup	UserService and User object is initialized.
Input	Object user (an instance of class User): • user.username: “david@yahoo.com” • user.password: “12345678” • user.type: “4”
Expected Output	Return a User object: • user.userId: -1

Test Case ID	UNIT02_HMTS_Login05
Purpose	Verify method UserService.Login can deal with exceptions when enter invalid user type data.
Test Setup	UserService and User object is initialized.
Input	Object user (an instance of class User): • user.username: “mary@fiu.edu” • user.password: “mynnnn” • user.type: “-1”
Expected Output	Return a User object: • user.userId: -1

Test Case ID	UNIT03_HMTS_Logout01
Purpose	Verify method UserService.logout can logged the user out of the system
Test Setup	UserService and User object are initialized.
Input	Pass UserID as a parameter: • user.userId: 2
Expected Output	true

Test Case ID	UNIT03_HMTS_Logout02
Purpose	Verify method UserService.logout with invalid user id.
Test Setup	UserService and User object are initialized.
Input	Pass user id as a parameter: • user.userId: 12
Expected Output	false

Test Case ID	UNIT04_HMTS_RolesControl01
Purpose	Verify method UserService.rolesControl can return a list of functions according to the role of the actor.
Test Setup	UserService and User object is initialized.
Input	Object user (an instance of class User): • user.userId: 1 • user.username: “Tony” • user.password: “123456” • user.firstName: “Tony” • user.lastName: “Smith” • user.phone: “786-000-0000” • user.role: “Manager” • user.secQuestionId: 1 • user.secAnswer: “Tony” • user.tag: “XYZ123”
Expected Output	“1,2,3,”

Test Case ID	UNIT04_HMTS_RolesControl02
Purpose	Verify method UserService.rolesControl can deal with exceptions when enter invalid data.
Test Setup	UserService and User object is initialized.
Input	Object user (an instance of class User): • user.userId: 1 • user.username: “Tony” • user.password: “123456” • user.firstName: “Tony” • user.lastName: “Smith” • user.phone: “786-000-0000” • user.role: “” • user.secQuestionId: 1 • user.secAnswer: “Tony” • user.tag: “XYZ123”
Expected Output	“”

Test Case ID	UNIT05_HMTS_VisualizeProducts01
Purpose	Verify method OperationService.displayProducts can display the list of products.
Test Setup	OperationService and Product object is initialized. List of products in the database: • ProductID = “1” • ProductName = “Turkey Breast” • ProductType = “Entrée” • ProductPrice = “8.99” • ProductDesc = “Roasted with a flavorful combination of butter and seasonings.” • ProductID = “2” • ProductName = “Vegetables” • ProductType = “Side” • ProductPrice = “2.99” • ProductDesc = “Chooped green pepper, onion, pimiento, corn, green beans, and green peas.” • ProductID = “3” • ProductName = “Lemonade” • ProductType = “Drink” • ProductPrice = “1.99”

	• ProductDesc = “One of the most enjoyable summer drinks; it refreshes and quenches thirst.”
Input	N/A
Expected Output	Object productView (an instance of class ModelAndView): • ProductID = “1” • ProductName = “Turkey Breast” • ProductType = “Entrée” • ProductPrice = “8.99” • ProductDesc = “Roasted with a flavorful combination of butter and seasonings.” • ProductID = “2” • ProductName = “Vegetables” • ProductType = “Side” • ProductPrice = “2.99” • ProductDesc = “Chooped green pepper, onion, pimiento, corn, green beans, and green peas.” • ProductID = “3” • ProductName = “Lemonade” • ProductType = “Drink” • ProductPrice = “1.99” • ProductDesc = “One of the most enjoyable summer drinks; it refreshes and quenches thirst.”

Test Case ID	UNIT05_HMTS_VisualizeProducts02
Purpose	Verify method OperationService.displayProducts can display the list of products when no product in database.
Test Setup	OperationService and Product object is initialized. There is no product in the database.
Input	N/A
Expected Output	The size of product list is 0.

Test Case ID	UNIT06_HMTS_AddProduct01
Purpose	Verify method OperationService.createProduct can add a new product to the system successfully.
Test Setup	OperationService and Product object is initialized.
Input	Object product (an instance of class Product): • product.name: “Pizza” • product.type: “Side” • product.price: 10.99 • product.brief: “Pork, Onions and Cheese.”
Expected Output	1

Test Case ID	UNIT06_HMTS_AddProduct02
Purpose	Verify method OperationService.createProduct can deal with exceptions when enter invalid data.
Test Setup	OperationService and Product object is initialized.
Input	Object product (an instance of class Product): • product.name: “” • product.type: “Side” • product.price: 10.99 • product.brief: “Pork, Onions and Cheese.”
Expected Output	0

Test Case ID	UNIT06_HMTS_AddProduct03
Purpose	Verify method OperationService.createProduct can deal with exceptions when enter invalid data.
Test Setup	OperationService and Product object is initialized.
Input	Object product (an instance of class Product): • product.name: “Pizza” • product.type: “” • product.price: 10.99 • product.brief: “Pork, Onions and Cheese.”
Expected Output	0

Test Case ID	UNIT06_HMTS_AddProduct04
Purpose	Verify method OperationService.createProduct can deal with exceptions when enter invalid data.
Test Setup	OperationService and Product object is initialized.
Input	Object product (an instance of class Product): • product.name: “Pizza” • product.type: “Side” • product.price: null • product.brief: “Pork, Onions and Cheese.”
Expected Output	0

Test Case ID	UNIT07_HMTS_VisualizeMenu01
Purpose	Verify method ServiceService.displayMenu can display the Menu
Test Setup	ServiceService and object products is initialized. List of products in the database: • ProductID = “1” • ProductName = “Turkey Breast” • ProductType = “Entrée” • ProductPrice = “8.99” • ProductDesc = “Roasted with a flavorful combination of butter and seasonings.” • ProductID = “2” • ProductName = “Vegetables” • ProductType = “Side” • ProductPrice = “2.99” • ProductDesc = “Chooped green pepper, onion, pimiento, corn, green beans, and green peas.” • ProductID = “3” • ProductName = “Lemonade” • ProductType = “Drink” • ProductPrice = “1.99” • ProductDesc = “One of the most enjoyable summer drinks; it refreshes and quenches thirst.”
Input	N/A
Expected Output	• ProductName = “Vegetables” • ProductType = “Side” • ProductPrice = “2.99” • ProductDesc = “Chooped green pepper, onion, pimiento, corn,

	green beans, and green peas.” • ProductName = “Turkey Breast” • ProductType = “Entrée” • ProductPrice = “8.99” • ProductDesc = “Roasted with a flavorful combination of butter and seasonings.” • ProductName = “Lemonade” • ProductType = “Drink” • ProductPrice = “1.99” • ProductDesc = “One of the most enjoyable summer drinks; it refreshes and quenches thirst.”

Test Case ID	UNIT07_HMTS_VisualizeMenu02
Purpose	Verify method ServiceService.displayMenu can display the Menu
Test Setup	ServiceService and object products are initialized. There is no product in the database:
Input	N/A
Expected Output	The size of product list is 0.

Test Case ID	UNIT08_HMTS_SelectProduct01
Purpose	Verify method ServiceService.selectProduct can add a new product to a customer’s shopping cart successfully.
Test Setup	ServiceService, Product and SelProduct object are initialized.
Input	Object product (an instance of class Product): • product.id: 1 • product.name: “Pizza” • product.type: “Side” • product.price: 10.99 • product.brief: “Pork, Onions and Cheese.” Object selproduct (an instance of class Product): • selProduct.quantity: 1 • selProduct.total: 10.99 • selProduct.userId: 3
Expected Output	1

Test Case ID	UNIT08_HMTS_SelectProduct02
Purpose	Verify method ServiceService.selectProduct can reject adding a new product to a customer's shopping cart without product id.
Test Setup	ServiceService, Product and SelProduct object are initialized.
Input	Object product (an instance of class Product): • product.id: *NULL* • product.name: "Pizza" • product.type: "Side" • product.price: 10.99 • product.brief: "Pork, Onions and Cheese." Object selproduct (an instance of class Product): • selProduct.quantity: 1 • selProduct.total: 10.99 • selProduct.userId: 3
Expected Output	0

Test Case ID	UNIT08_HMTS_SelectProduct03
Purpose	Verify method ServiceService.selectProduct can reject adding a new product to a customer's shopping cart without product quantity.
Test Setup	ServiceService, Product and SelProduct object are initialized.
Input	Object product (an instance of class Product): • product.id: 1 • product.name: "Pizza" • product.type: "Side" • product.price: 10.99 • product.brief: "Pork, Onions and Cheese." Object selproduct (an instance of class Product): • selProduct.quantity: *NULL* • selProduct.total: 10.99 • selProduct.userId: 3
Expected Output	0

Test Case ID	UNIT08_HMTS_SelectProduct04
Purpose	Verify method ServiceService.selectProduct can reject adding a new product to a customer's shopping cart without user id.
Test Setup	ServiceService, Product and SelProduct object are initialized.
Input	Object product (an instance of class Product): • product.id: 1 • product.name: "Pizza" • product.type: "Side" • product.price: 10.99 • product.brief: "Pork, Onions and Cheese." Object selproduct (an instance of class Product): • selProduct.quantity: 1 • selProduct.total: 10.99 • selProduct.userId: *NULL*
Expected Output	0

Test Case ID	UNIT09_HMTS_ViewCart01
Purpose	Verify method ServiceService.displayCart can return a list of SelProduct representing the products selected by the actor.
Test Setup	ServiceService and userid is initialized. The cart of current user (userId: "2") contains: Pizza (productId: 1, userId: 2, quantity: 1), Coke (productId: 2, userId: 2, quantity: 2).
Input	N/A
Expected Output	A list contains two SelProduct. The first one's productId is "1", userId is "2" and quantity is "1". The second one's productId is "2", userId is "2" and quantity is "2".

Test Case ID	UNIT09_HMTS_ViewCart02
Purpose	Verify method ServiceService.displayCart can deal with exceptions when there is invalid data.
Test Setup	ServiceService and userid is initialized. The cart of current user (userId: "1") contains: Pizza (productId: null, userId: 2, quantity: 1).
Input	N/A
Expected Output	An empty list of SelProduct.

Test Case ID	UNIT09_HMTS_ViewCart03
Purpose	Verify method ServiceService.displayCart can deal with exceptions when there is invalid data.
Test Setup	ServiceService and userid is initialized. The cart of current user (userId: "1") contains: Pizza (productId: 1, userId: null, quantity: 1).
Input	N/A
Expected Output	An empty list of SelProduct.

Test Case ID	UNIT09_HMTS_ViewCart04
Purpose	Verify method ServiceService.displayCart can deal with exceptions when there is invalid data.
Test Setup	ServiceService and userid is initialized. The cart of current user (userId: "1") contains: Pizza (productId: 1, userId: 2, quantity: null).
Input	N/A
Expected Output	An empty list of SelProduct.

Test Case ID	UNIT10_HMTS_RemoveProduct01
Purpose	Testing operation of ServiceService.delProductCart function.
Test Setup	ServiceService is initialized. A product exists in the cart of a particular customer.
Input	Productid: 001 Userid: 001
Expected Output	1

Test Case ID	UNIT10_HMTS_RemoveProduct02
Purpose	Testing operation of ServiceService.delProductCart function.
Test Setup	ServiceService is initialized. A product does not exist for a particular customer.
Input	Productid: 002 Userid: 001
Expected Output	Exception

Test Case ID	UNIT11_HMTS_FillOutOrderInfo01
Purpose	Verify method ServiceService.PlaceOrder() can place a cash order successfully.
Test Setup	order, payment, orderProduct are initialized
Input	Object order (an instance of class Order): • Phone: 305 474 2378 • Shipping Address: 123 NW 14th St, Miami FL 33100 • Status: processing • User ID: 3L Object payment (an instance of class Payment): • Amount: $8.99 • Method: cash Object orderProduct (an instance of class OrderProduct): • Name: Pizza • Type: Side • Price: $8.99 • Quantity: 1 • Amount: $8.99
Expected Output	1

Test Case ID	UNIT11_HMTS_FillOutOrderInfo02
Purpose	Verify method ServiceService.PlaceOrder() does not place an invalid cash order.
Test Setup	order, payment, orderProduct are initialized
Input	Object order (an instance of class Order): • Phone: 305 474 2378 • Shipping Address: 123 NW 14th St, Miami FL 33100 • Status: processing • User ID: null Object payment (an instance of class Payment): • Amount: $8.99 • Method: cash Object orderProduct (an instance of class OrderProduct): • Name: Pizza • Type: Side • Price: $8.99 • Quantity: 1

	• Amount: $8.99
Expected Output	0

Test Case ID	UNIT11_HMTS_FillOutOrderInfo03
Purpose	Verify method ServiceService.PlaceOrder() does not place a cash order with invalid products.
Test Setup	payment, orderProduct are initialized
Input	Object payment (an instance of class Payment): • Amount: $8.99 • Method: cash Object orderProduct (an instance of class OrderProduct): • Name: Pizza • Type: Side • Price: $8.99 • Quantity: 1 • Amount: $8.99
Expected Output	0

Test Case ID	UNIT11_HMTS_FillOutOrderInfo04
Purpose	Verify method ServiceService.PlaceOrder() does not place a cash order with no products list specified.
Test Setup	order, payment are initialized
Input	Object order (an instance of class Order): • Phone: 305 474 2378 • Shipping Address: 123 NW 14th St, Miami FL 33100 • Status: processing • User ID: null Object payment (an instance of class Payment): • Amount: $8.99 • Method: cash
Expected Output	0

Test Case ID	UNIT11_HMTS_FillOutOrderInfo05
Purpose	Verify method ServiceService.PlaceOrder() does not place a cash order with invalid products list.
Test Setup	order, payment, orderProduct are initialized.

Input	Object order (an instance of class Order): • Phone: 305 474 2378 • Shipping Address: 123 NW 14th St, Miami FL 33100 • Status: processing • User ID: null Object payment (an instance of class Payment): • Amount: $8.99 • Method: cash Object orderProduct (an instance of class OrderProduct): • Name: null • Type: Side • Price: $8.99 • Quantity: 1 • Amount: $8.99
Expected Output	0

Test Case ID	UNIT11_HMTS_FillOutOrderInfo06
Purpose	Verify method ServiceService.PlaceOrder() does not place a cash order with invalid payment method.
Test Setup	order, payment, orderProduct are initialized
Input	Object order (an instance of class Order): • Phone: 305 474 2378 • Shipping Address: 123 NW 14th St, Miami FL 33100 • Status: processing • User ID: 3L Object payment (an instance of class Payment): • Amount: $8.99 • Method: null Object orderProduct (an instance of class OrderProduct): • Name: Pizza • Type: Side • Price: $8.99 • Quantity: 1 • Amount: $8.99
Expected Output	0

Test Case ID	UNIT11_HMTS_FillOutOrderInfo07
Purpose	Verify method ServiceService.PlaceOrder() can place a card order successfully.
Test Setup	card, order, payment, orderProduct are initialized
Input	Object card (an instance of class Card): • Card number: 37283937283737283 • Name: John David • Expiration Date: 05/2020 • Security Code: 876 • Billing Address: 123 NW 14th St, Miami FL 33100 Object order (an instance of class Order): • Phone: 305 474 2378 • Shipping Address: 123 NW 14th St, Miami FL 33100 • Status: processing • User ID: 3L Object payment (an instance of class Payment): • Amount: $8.99 • Method: card Object orderProduct (an instance of class OrderProduct): • Name: Pizza • Type: Side • Price: $8.99 • Quantity: 1 • Amount: $8.99
Expected Output	1

Test Case ID	UNIT11_HMTS_FillOutOrderInfo08
Purpose	Verify method ServiceService.PlaceOrder() does not place a card order with invalid card information.
Test Setup	card, order, payment, orderProduct are initialized
Input	Object card (an instance of class Card): • Card number: null • Name: John David • Expiration Date: 05/2020 • Security Code: 876 • Billing Address: 123 NW 14th St, Miami FL 33100 Object order (an instance of class Order):

	• Phone: 305 474 2378 • Shipping Address: 123 NW 14th St, Miami FL 33100 • Status: processing • User ID: 3L Object payment (an instance of class Payment): • Amount: $8.99 • Method: card Object orderProduct (an instance of class OrderProduct): • Name: Pizza • Type: Side • Price: $8.99 • Quantity: 1 Amount: $8.99
Expected Output	0

7.2. Subsystem Tests

In this section, we describe the subsystem tests of our system. The test cases are shown in the tables below.

Test Case ID	SUBSYS01_HMTS_CreateAccount01
Purpose	Testing operation of UserController.register function. Show that subsystem can add a new user for web service.
Test Setup	UserController is runing.
Input	Pass multiple attributes as parameters: • mode: “register” • firstName: “Bob” • lastName: “Hope” • username: “bobhope@gmail.com” • password: “bobhope13” • phone: “(213)112-2432” • role: 2
Expected Output	Status: OK View.name: “users” Attrubutes.userId: 3 Attrubutes.firstName: “Bob” Attrubutes.role: 2 Attrubutes.func: “1,2,”

Test Case ID	SUBSYS01_HMTS_CreateAccount02
Purpose	Testing operation of UserController.register function. Show that subsystem can add a new user with invalid username for web service.
Test Setup	UserController is runing.
Input	Pass multiple attributes as parameters: • mode: “register” • firstName: “Bob” • lastName: “Hope” • username: “” • password: “bobhope13” • phone: “(213)112-2432” • role: 2
Expected Output	Status: OK Result: “failed” Attrubutes.userId: *NULL* Attrubutes.firstName: *NULL* Attrubutes.role: *NULL*

Test Case ID	SUBSYS01_HMTS_CreateAccount03
Purpose	Testing operation of UserController.register function. Show that subsystem can add a new user with invalid mode value for web service.
Test Setup	UserController is runing.
Input	Pass multiple attributes as parameters: • mode: “registeration” • firstName: “Bob” • lastName: “Hope” • username: “bobhope@gmail.com” • password: “bobhope13” • phone: “(213)112-2432” • role: 2
Expected Output	Status: 404

Test Case ID	SUBSYS01_HMTS_CreateAccount04
Purpose	Testing operation of UserController.register function. Show that subsystem can add a new user for mobile service.
Test Setup	UserController is runing.
Input	Pass multiple attributes as parameters: • mode: “register” • firstName: “Bob” • lastName: “Hope” • username: “bobhope@gmail.com” • password: “bobhope13” • phone: “(213)112-2432” • role: 4
Expected Output	Status: OK View.name: “users” Attrubutes.userId: 3 Attrubutes.firstName: “Bob” Attrubutes.role: 4

Test Case ID	SUBSYS01_HMTS_CreateAccount05
Purpose	Testing operation of UserController.register function with invalid username for mobile service.
Test Setup	UserController is runing.
Input	Pass multiple attributes as parameters: • mode: “register” • firstName: “Bob” • lastName: “Hope” • username: “” • password: “bobhope13” • phone: “(213)112-2432” • role: 4
Expected Output	Status: OK Result: “failed” Attrubutes.userId: *NULL* Attrubutes.firstName: *NULL* Attrubutes.role: *NULL*

Test Case ID	SUBSYS01_HMTS_CreateAccount06
Purpose	Testing operation of UserController.register function. Show that subsystem can add a new user with invalid mode value for mobile service.
Test Setup	UserController is runing.
Input	Pass multiple attributes as parameters: • mode: “registeration” • firstName: “Bob” • lastName: “Hope” • username: “bobhope@gmail.com” • password: “bobhope13” • phone: “(213)112-2432” • role: 2
Expected Output	Status: 404

Test Case ID	SUBSYS02_HMTS_Login01
Purpose	Verify interface UserController.Login can login into the system successfully for web service.
Test Setup	UserController is runing.
Input	Pass multiple attributes as parameters: • mode: “login” • username: “mary@fiu.edu” • password: “my2882”
Expected Output	Status: OK View.name: “home” Username: “mary@fiu.edu” Password: “my2882” Type: “2” Func: “1,2,”

Test Case ID	SUBSYS02_HMTS_Login02
Purpose	Verify interface UserController.Login can deal with exceptions when enter invalid password for web service.
Test Setup	UserController is runing.
Input	Pass multiple attributes as parameters: • mode: “login” • user.username: “nick” • user.password: “my0000”
Expected Output	Status: OK View.name: “login”

Test Case ID	SUBSYS02_HMTS_Login03
Purpose	Verify interface UserController.Login can login into the system successfully for mobile service.
Test Setup	UserController is runing.
Input	Pass multiple attributes as parameters: • mode: “login” • user.username: “david@yahoo.com” • user.password: “123456”
Expected Output	Status: OK View.name: “home” Username: “david@yahoo.com” Type: “4”

Test Case ID	SUBSYS02_HMTS_Login04
Purpose	Verify interface UserController.Login can deal with exceptions when enter invalid password for mobile service.
Test Setup	UserController is runing.
Input	Pass multiple attributes as parameters: • mode: “login” • user.username: “david@yahoo.com” • user.password: “12345678”
Expected Output	Status: OK View.name: “home”

Test Case ID	SUBSYS02_HMTS_Login05
Purpose	Verify interface UserController.Login can deal with exceptions when enter invalid mode value.
Test Setup	UserController is runing.
Input	Pass multiple attributes as parameters: • mode: “log_in” • user.username: “mary@fiu.edu” • user.password: “mynnnn”
Expected Output	Status: 404

Test Case ID	SUBSYS03_HMTS_Logout01
Purpose	Verify interface UserController.Login can logged the user out of the system
Test Setup	UserController is running.
Input	N/A
Expected Output	true

Test Case ID	SUBSYS04_HMTS_VisualizeProducts01
Purpose	Verify interface OperationController.displayProducts can return a view of product list.
Test Setup	OperationController and HttpServletRequest object is initialized. The size of a product list is 3.
Input	Object request (which is an instance of class HttpServletRequest): “Show Product List”
Expected Output	Object productView (an instance of class ModelAndView): productView.viewName: “products” productView.Objects.proList.size: 3

Test Case ID	SUBSYS04_HMTS_VisualizeProducts02
Purpose	Verify interface OperationController.displayProducts can return a view of product list.
Test Setup	OperationController and HttpServletRequest object is initialized. The size of a product list is 0.
Input	Object request (which is an instance of class HttpServletRequest): “Show Product List”
Expected Output	Object productView (an instance of class ModelAndView):

	productView.viewName: “products” productView.Objects.proList.size: 0

Test Case ID	SUBSYS05_HMTS_AddProduct01
Purpose	Verify interface OperationController.createProduct can add a new product to the system successfully and return a view of product list.
Test Setup	OperationController and HttpServletRequest object is initialized. The size of a product list is 0.
Input	Object request (which is an instance of class HttpServletRequest): • request.parameters.name: “Pizza” • request.parameters.type: “Side” • request.parameters.price: 10.99 • request.parameters.brief: “Pork, Onions and Cheese.”
Expected Output	Object productView (an instance of class ModelAndView): • productView.viewName: “products”; • productView.Objects.proList.size: 1; • proList[0].name: “Pizza”; • proList[0].type: “Side”; • proList[0].type: 10.99; • proList[0].brief: “Pork, Onions and Cheese.”.

Test Case ID	SUBSYS05_HMTS_AddProduct02
Purpose	Verify interface OperationController.createProduct can deal with exceptions when enter invalid data, then return a view of product list.
Test Setup	OperationController and HttpServletRequest object is initialized. The size of a product list is 0.
Input	Object request (which is an instance of class HttpServletRequest): • request.parameters.name: “” • request.parameters.type: “Side” • request.parameters.price: 10.99 • request.parameters.brief: “Pork, Onions and Cheese.”
Expected Output	Object productView (an instance of class ModelAndView): • productView.viewName: “products”; • productView.Objects.proList.size: 0.

Test Case ID	SUBSYS05_HMTS_AddProduct03
Purpose	Verify interface OperationController.createProduct can deal with exceptions when enter invalid data, then return a view of product list.
Test Setup	OperationController and HttpServletRequest object is initialized. The size of a product list is 0.
Input	Object request (which is an instance of class HttpServletRequest): • request.parameters.name: “Pizza” • request.parameters.type: “” • request.parameters.price: 10.99 • request.parameters.brief: “Pork, Onions and Cheese.”
Expected Output	Object productView (an instance of class ModelAndView): • productView.viewName: “products”; • productView.Objects.proList.size: 0.

Test Case ID	SUBSYS06_HMTS_VisualizeMenu01
Purpose	Verify interface MobileInterface.displayMenu can return a view of the menu.
Test Setup	OperationInterface and map object is initialized. The size of a product list is 3.
Input	Object request: “Show Menu”
Expected Output	Object map (an instance of class Map): map.Result: “succesful” map.Count: “3” map.Data: “prodList”

Test Case ID	SUBSYS06_HMTS_VisualizeMenu02
Purpose	Verify interface MobileInterface.displayMenu can return a view of the menu.
Test Setup	OperationInterface and map object is initialized. The size of a product list is 0.
Input	Object request: “Show Menu”
Expected Output	Object map (an instance of class Map): map.Result: “succesful” map.Count: “0” map.Data: “prodList”

Test Case ID	SUBSYS07_HMTS_SelectProduct01
Purpose	Verify interface MobileInterface.selectProduct can add a new product to a customer's shopping cart successfully.
Test Setup	MobileInterface is initialized.
Input	Pass multiple attributes as parameters: • id: 1 • name: "Pizza" • type: "Side" • price: 10.99 • brief: "Pork, Onions and Cheese." • quantity: 1 • total: 10.99 • userId: 3
Expected Output	1

Test Case ID	SUBSYS07_HMTS_SelectProduct02
Purpose	Verify interface MobileInterface.selectProduct can reject adding a new product to a customer's shopping cart without product id.
Test Setup	MobileInterface is initialized.
Input	Pass multiple attributes as parameters: • id: *NULL* • name: "Pizza" • type: "Side" • price: 10.99 • brief: "Pork, Onions and Cheese." • quantity: 1 • total: 10.99 • userId: 3
Expected Output	0

Test Case ID	SUBSYS07_HMTS_SelectProduct03
Purpose	Verify interface MobileInterface.selectProduct can reject adding a new product to a customer's shopping cart without product quantity.
Test Setup	MobileInterface is initialized.
Input	Pass multiple attributes as parameters:

	• id: 1 • name: “Pizza” • type: “Side” • price: 10.99 • brief: “Pork, Onions and Cheese.” • quantity: *NULL* • total: 10.99 • userId: 3
Expected Output	0

Test Case ID	SUBSYS07_HMTS_SelectProduct04
Purpose	Verify interface MobileInterface.selectProduct can reject adding a new product to a customer’s shopping cart without user id.
Test Setup	MobileInterface is initialized.
Input	Pass multiple attributes as parameters: • id: 1 • name: “Pizza” • type: “Side” • price: 10.99 • brief: “Pork, Onions and Cheese.” • quantity: 1 • total: 10.99 • userId: *NULL*
Expected Output	0

Test Case ID	SUBSYS08_HMTS_ViewCart01
Purpose	Verify interface ServiceController.displayCart can return a view of the cart containing selected products.
Test Setup	ServiceController and HttpServletRequest object is initialized. The size of the SelProduct list of the cart whose owner’s user id is 3 is 2.
Input	Object request (which is an instance of class HttpServletRequest): UserID: 3
Expected Output	Object cartView (an instance of class ModelAndView): cartView.viewName: “cart” cartView.Objects.selProList.size: 2 The first one’s productId is “1”, userId is “2” and quantity is “1”. The second one’s productId is “2”, userId is “2” and quantity is “2”.

Test Case ID	SUBSYS08_HMTS_ViewCart02
Purpose	Verify interface ServiceController.displayCart can return a view of the cart containing selected products.
Test Setup	ServiceController and HttpServletRequest object is initialized. The size of the SelProduct list of the cart whose owner's user id is 4 is 0.
Input	Object request (which is an instance of class HttpServletRequest): UserID: 4
Expected Output	Object cartView (an instance of class ModelAndView): cartView.viewName: "cart" cartView.Objects.selProList.size: 0

Test Case ID	SUBSYS08_HMTS_ViewCart03
Purpose	Verify interface ServiceController.displayCart can return a view of the cart containing selected products.
Test Setup	ServiceController and HttpServletRequest object is initialized.
Input	Object request (which is an instance of class HttpServletRequest): Invalid user id: -1
Expected Output	Object cartView (an instance of class ModelAndView): cartView.viewName: "cart" cartView.Objects.selProList.size: 0

Test Case ID	SUBSYS09_HMTS_RemoveProduct01
Purpose	Testing interface ServiceController.delProductCart function. Show that subsystem can remove product from cart and take customer back to the original cart.
Test Setup	ServiceController is initialized. A product exists in the cart of a particular customer.
Input	Productid: 001 Userid: 001
Expected Output	The shopping cart is empty (the size of select product list is 0).

Test Case ID	SUBSYS09_HMTS_RemoveProduct01
Purpose	Testing interface ServiceController.delProductCart function. Show that subsystem can remove product from cart and take customer back to the original cart.
Test Setup	ServiceController is initialized. A product exists in the cart of a particular customer.
Input	Productid: 1 Userid: 3
Expected Output	The shopping cart is empty (the size of select product list is 0).

Test Case ID	SUBSYS09_HMTS_RemoveProduct02
Purpose	Testing interface ServiceController.delProductCart function. Show that subsystem can remove product from cart and take customer back to the original cart.
Test Setup	ServiceController is initialized. No product exists in the cart of a particular customer.
Input	Productid: 1 Userid: 3
Expected Output	The shopping cart is empty (the size of select product list is 0).

Test Case ID	SUBSYS10_HMTS_FillOutOrderInfo01
Purpose	Verify method ServiceController.newOrder can return a view of the order containing the detail information.
Test Setup	ServiceController and HttpServletRequest object is initialized.
Input	Object request (which is an instance of class HttpServletRequest): request.parameters.shippingAddress: “123 Main Street, Toad Town, SM, 12345” request.parameters.phoneNumber: “” request.parameters.deliveryNotes: “Please use doorbell”
Expected Output	Object orderView (an instance of class ModelAndView): orderView.viewName: “order” orderView.Objects.order.shippingAddress: “123 Main Street, Toad Town, SM, 12345” orderView.Objects.order.phoneNumber: “” orderView.Objects.order.deliveryNotes: “Please use doorbell”

Test Case ID	SUBSYS10_HMTS_FillOutOrderInfo02
Purpose	Verify method ServiceController.newOrder can deal with exceptions when entering invalid data.
Test Setup	ServiceController and HttpServletRequest object is initialized.
Input	Object request (which is an instance of class HttpServletRequest): request.parameters.shippingAddress: “” request.parameters.phoneNumber: “” request.parameters.deliveryNotes: “”
Expected Output	Object orderView (an instance of class ModelAndView): orderView.viewName: “order” orderView.Objects.order: null

7.3. System Tests

In this section, we describe the system tests of our system. The test cases are shown in the tables below.

Test Case ID	SYS01_HMTS_CreateAccount01
Purpose	Testing functionality of creating a new user (Staff) in the system.
Test Setup	The system is running. Manager is logged into system. No staff info is in database.
Input	User.firstName: “Steven” User.lastName: “McQueen” User.username: “steve@gmail.com” User.password: “fast” User.phone: (213)112-2432 User.role: Staff
Expected Output	Show user page to manager. • User.firstName: “Steven” • User.lastName: “McQueen” • User.username: “steve@gmail.com” • User.password: “fast” • User.phone: (213)112-2432 • User.role: Staff

Test Case ID	SYS01_HMTS_CreateAccount02
Purpose	Testing functionality of creating a new user (customer) in the system.
Test Setup	The system is running. There is no active login in the session.
Input	User.firstName: “Bob” User.lastName: “Hope” User.username: “bobhope@gmail.com” User.password: “bobhope13” User.phone: (213)112-2432 User.role: customer
Expected Output	Show customer to landing page for placing orders and viewing the menu.

Test Case ID	SYS01_HMTS_CreateAccount03
Purpose	Testing functionality of creating a new user (customer) in the system. Rainy Day Scenario: User is already in the system.
Test Setup	The system is running. There is no active login in the session.
Input	User.firstName: “Bob” User.lastName: “Hope” User.username: “bobhope@gmail.com” User.password: “bobhope13” User.phone: (213)112-2432 User.role: customer
Expected Output	Shows error to customer that the user is already in the system. Shows login screen.

Test Case ID	SYS02_HMTS_Login01
Purpose	Testing functionality of logging into the system. Test using correct credentials
Test Setup	The system is running. There is no active login in the session.
Input	User.username: “bobhope@gmail.com” User.password: “bobhope13”
Expected Output	1. (takes user to ordering page).

Test Case ID	SYS02_HMTS_Login02
Purpose	Testing functionality of logging into the system. Test using incorrect credentials
Test Setup	UserController is initialized. There is no active login in the session.
Input	User.username: “bobhope@gmail.com” User.password: “bobhope14”
Expected Output	Incorrect login message is displayed (takes user back to login page).

Test Case ID	SYS02_HMTS_Login03
Purpose	Testing functionality of logging into the system. Test using incomplete credentials
Test Setup	The system is running. There is no active login in the session.
Input	User.username: “bobhope@gmail.com” User.password:
Expected Output	Error thrown for incomplete arguments.

Test Case ID	SYS03_HMTS_Logout01
Purpose	Verify the system can log out a registered, logged-in user.
Test Setup	Joe has a valid customer account and has logged in into the system.
Input	Joe clicks on the “Log Out” button.
Expected Output	System displays the log in page.

Test Case ID	SYS03_HMTS_Logout02
Purpose	Verify the system can log out a registered, logged-in user.
Test Setup	Caroline has a valid staff account and has logged in into the system.
Input	Caroline clicks on the “Log Out” button.
Expected Output	System displays the log in page.

Test Case ID	SYS03_HMTS_Logout03
Purpose	Verify the system can refrain from logging out a user if there is an inconsistency.
Test Setup	John has a valid customer account and has logged in into the system. John added some products to his shopping cart and is checking out.
Input	Instead of clicking the “Place Order” button, John clicks on “Log Out”.
Expected Output	Pop a warning: “If you log out now, your order will not be placed!”

Test Case ID	SYS04_HMTS_RolesControl01
Purpose	Verify the system can show all the functions an actor has access to after he/she signs in the system.
Test Setup	Tony has entered the “Login” web page. His role is Manager in the system.
Input	User Name: Tony Password: 123456
Expected Output	Show a function list of all Manager’s functions which are available on homepage, including “Manage staff accounts”, “Manage products”, and “Manage orders”.

Test Case ID	SYS04_HMTS_RolesControl02
Purpose	Verify the system can show all the functions an actor has access to after he/she signs in the system.
Test Setup	Tom has entered the “Login” web page. His role is Deliveryman in the system.
Input	User Name: Tom Password: 000000
Expected Output	Show a function list of all Deliveryman’s function(s) available on homepage, including only one item: “Visualize orders”.

Test Case ID	SYS04_HMTS_RolesControl03
Purpose	Verify the system can reject showing the functions when entering invalid data.
Test Setup	Tom has entered the “Login” web page. His role is Deliveryman in the system.
Input	User Name: Tom Password: 111111
Expected Output	Pop a warning “Invalid user name or password”.

Test Case ID	SYS05_HMTS_VisualizeProducts01
Purpose	Verify the system can display the list of products successfully.
Test Setup	Mary has logged into the system and entered the “Product” web page.
Input	N/A
Expected Output	Show a product list on which there is only one product: • Product Name: “Turkey breast” • Product Type: “Entree” • Product Price: “10.99” • Product Brief: “Roasted with a flavorful combination of butter and seasonings.”

Test Case ID	SYS05_HMTS_VisualizeProducts02
Purpose	Verify the system can display the list of products successfully.
Test Setup	Mary has logged into the system and entered the “Product” web page.
Input	N/A
Expected Output	Show a product list on which there is two products: • Product Name: “Turkey breast” • Product Type: “Entree” • Product Price: “10.99” • Product Brief: “Roasted with a flavorful combination of butter and seasonings.” • Product Name = “Vegetables” • Product Type = “Side” • Product Price = “2.99” • Product Brief = “Chooped green pepper, onion, pimiento, corn, green beans, and green peas.”

Test Case ID	SYS05_HMTS_VisualizeProducts03
Purpose	Verify the system can display the list of products empty.
Test Setup	Mary has logged into the system and entered the “Product” web page.
Input	N/A
Expected Output	Show a product list which is empty: • Product Name: “ ” • Product Type: “ ”

	• Product Price: “ ” • Product Brief: “ ”

Test Case ID	SYS06_HMTS_AddProduct01
Purpose	Verify the system can add a new product to the system successfully.
Test Setup	Jack has logged into the system and entered the “Product” web page, while there is no product on a product list.
Input	Fill out the fields of a new product: • Product Name: Pizza • Product Type: Side • Product Price: 10.99 • Product Brief: Pork, Onions and Cheese.
Expected Output	Show a product list on which there is only one product: • Product Name: Pizza; • Product Type: Side; • Product Price: 10.99; • Product Brief: Pork, Onions and Cheese.

Test Case ID	SYS06_HMTS_AddProduct02
Purpose	Verify the system can add a new product to the system successfully.
Test Setup	Jack has logged into the system and entered the “Product” web page, while there is no product on a product list.
Input	Fill out the fields of a new product: • Product Name: Lemonade • Product Type: Drink • Product Price: 1.99 • Product Brief: NULL
Expected Output	Show a product list on which there is only one product: • Product Name: Lemonade • Product Type: Drink • Product Price: 1.99 • Product Brief: (blank)

Test Case ID	SYS06_HMTS_AddProduct03
Purpose	Verify the system can reject inserting a new product when enter invalid data.

Test Setup	Jack has logged into the system and entered the "Product" web page, while there is no product on a product list.
Input	Fill out the fields of a new product: • Product Name: NULL • Product Type: Side • Product Price: 10.99 • Product Brief: Pork, Onions and Cheese.
Expected Output	Pop a warning "The information is invalid".

Test Case ID	SYS07_HMTS_VisualizeMenu01
Purpose	Verify the system can display the menu successfully.
Test Setup	Mary has logged into the system and entered the "Menu" page.
Input	N/A
Expected Output	Show a product list on which there is only one product: • Product Name = "Vegetables" • Product Type = "Side" • Product Price = "2.99" • Product Brief = "Chooped green pepper, onion, pimiento, corn, green beans, and green peas."

Test Case ID	SYS07_HMTS_VisualizeMenu02
Purpose	Verify the system can display the list of products successfully.
Test Setup	Mary has logged into the mobile app and entered the "Menu" page.
Input	N/A
Expected Output	Show a product list on which there is two products: • Product Name = "Vegetables" • Product Type = "Side" • Product Price = "2.99" • Product Brief = "Chooped green pepper, onion, pimiento, corn, green beans, and green peas."Product Name: "Turkey breast" • Product Type: "Entree" • Product Price: "10.99" • Product Brief: "Roasted with a flavorful combination of butter and seasonings."

Test Case ID	SYS07_HMTS_VisualizeMenu03
Purpose	Verify the system can display the list of products empty.
Test Setup	Mary has logged into the system and entered the “Menu” page.
Input	N/A
Expected Output	Show a product list which is empty: • Product Name: “ ” • Product Type: “ ” • Product Price: “ ” • Product Brief: “ ”

Test Case ID	SYS08_HMTS_SelectProduct01
Purpose	Verify the system can add a new product to a customer’s shopping cart successfully.
Test Setup	The system is running. No product is in the shopping cart of Tom (user id is 3).
Input	Tom inputs a number 3 into a textbox that is following the price of “Pizze” product. Tom pushes the “Add” button that is following the quantity textbox.
Expected Output	The system saves the record of three products of “Pizza” into the cart of Tom. • Name: Pizza • Quantity: 3

Test Case ID	SYS08_HMTS_SelectProduct02
Purpose	Verify the system can add a new product to a customer’s shopping cart successfully.
Test Setup	The system is running. No product is in the shopping cart of Bob (user id is 4).
Input	Bob inputs a number 2 into a textbox that is following the price of “Vegetables” product. Bob pushes the “Add” button that is following the quantity textbox.
Expected Output	The system saves the record of three products of “Vegetable” into the cart of Bob. • Name: Vegetable • Quantity: 2

Test Case ID	SYS08_HMTS_SelectProduct03
Purpose	Verify the system can reject adding a product to a customer's shopping cart when the quantity is over 20.
Test Setup	The system is running. Twenty "Pizza" Products are in the shopping cart of Jack (user id is 2).
Input	Jack inputs a number 3 into a textbox that is following the price of "Pizze" product. Jack pushes the "Add" button that is following the quantity textbox.
Expected Output	The system doesn't save the product "Pizza" into the cart of Tom. • Name: Pizza • Quantity: 20

Test Case ID	SYS09_HMTS_ViewCart01
Purpose	Verify the system can show the cart that contains all the products selected by the actor.
Test Setup	Jack has logged into the system and added 1 Pizza into his cart.
Input	None.
Expected Output	Show the cart in which there is only one product whose name is "Pizza", type is "Side", price is "10.99" and quantity is "1".

Test Case ID	SYS09_HMTS_ViewCart02
Purpose	Verify the system can show the cart that contains all the products selected by the actor.
Test Setup	Tom has logged into the system and added 1 Pizza and 2 Coke into his cart.
Input	None.
Expected Output	Show the cart in which there are two products. The first one's name is "Pizza", type is "Side", price is "10.99" and quantity is "1". The second one's name is "Coke", type is "Drink", price is "1.99" and quantity is "2".

Test Case ID	SYS09_HMTS_ViewCart03
Purpose	Verify the system can show the cart when it is empty.
Test Setup	Tom has logged into the system and nothing is in his cart.
Input	None.
Expected Output	Show the cart in which there is no product.

Test Case ID	SYS10_HMTS_RemoveProduct01
Purpose	Testing functionality of removing a product from the cart.
Test Setup	Active user 01 is logged into system. User is in the cart view. There are two products in the cart (Pid: 01 and Pid: 02).
Input	User click to remove the product Pid: 02 from the cart.
Expected Output	System shows the cart with only one product (Pid: 01) left in it.

Test Case ID	SYS10_HMTS_RemoveProduct02
Purpose	Testing functionality of removing a product from the cart.
Test Setup	Active user 01 is logged into system. User is in the cart view. There is one product in the cart (Pid: 01).
Input	User click to remove the product Pid: 01 from the cart.
Expected Output	System shows an empty cart.

Test Case ID	SYS10_HMTS_RemoveProduct03
Purpose	Testing functionality of removing a product from the cart.
Test Setup	Active user 02 is logged into system. User is in the cart view. There is no product in the cart (Pid: 01).
Input	User click to remove the product Pid: 01 from the cart.
Expected Output	System shows an empty cart.

Test Case ID	SYS11_HMTS_FillOutOrderInfo01
Purpose	Verify the system can display a selection menu with payment methods, provide fields and buttons for chosen payment method, display correct fields for “cash” payment method, and request order placement.
Test Setup	John has logged in into the system, added some products to his shopping cart, and requested to check out. He has selected “Pay by cash”.
Input	Fill out the fields required by this payment form: • Shipping Address: 123 Main Street, Toad Town, SM, 12345 • Phone number (optional): blank • Delivery notes (optional): Please use doorbell. Click on “Place Order” button.
Expected Output	Show an order confirmation: • Shipping address: 123 Main Street, Toad Town, SM, 12345 • Payment method: cash • Delivery note: “Please use doorbell.”

Test Case ID	SYS11_HMTS_FillOutOrderInfo02
Purpose	Verify the system can display a selection menu with payment methods, provide fields and buttons for chosen payment method, display correct fields for "card" payment method, verify address consistency, and request order placement.
Test Setup	John has logged in into the system, added some products to his shopping cart, and requested to check out. He has selected "Pay by debit/credit card".
Input	Fill out the fields required by this payment form: • Name on card: John Doe • Card number: 1234 5678 9123 4567 • Expiration date: 01/20 • Security code: 012 • Billing address: 123 Main Street, Toad Town, SM, 12345 • Shipping address: 123 Main Street, Toad Town, SM, 12345 • Phone number (optional): blank • Delivery notes (optional): Please use doorbell. Click on "Place Order" button.
Expected Output	Show an order confirmation: • Shipping address: 123 Main Street, Toad Town, SM, 12345 • Payment method: debit/credit card • Card ending: 4567 • Billing address: 123 Main Street, Toad Town, SM, 12345 • Delivery note: "Please use doorbell."

Test Case ID	SYS11_HMTS_FillOutOrderInfo03
Purpose	Verify the system can reject the request to place an order if there is an inconsistency in the input.
Test Setup	John has logged in into the system, added some products to his shopping cart, and requested to check out. He has selected "Pay by debit/credit card".
Input	Fill out the fields required by this payment form: • Name on card: John Doe • Card number: 1234 5678 9123 4567 • Expiration date: 01/20 • Security code: blank • Billing address: 123 Main Street, Toad Town, SM, 12345

	• Shipping address: 123 Main Street, Toad Town, SM, 12345 • Phone number (optional): blank • Delivery notes (optional): Please use doorbell. Click on “Place Order” button.
Expected Output	Pop a warning: “Security code missing!”

Test Case ID	SYS12_HMTS_PopAddressAlert01
Purpose	Verify the system can detect inconsistency between billing and shipping address and prompt the user for action.
Test Setup	John has logged in into the system, added some products to his shopping cart, and requested to check out. He has selected “Pay by debit/credit card”.
Input	Fill out the fields required by this payment form: • Name on card: John Doe • Card number: 1234 5678 9123 4567 • Expiration date: 01/20 • Security code: blank • Billing address: 123 Main Street, Treetop Town, SM, 12345 • Shipping address: 123 Main Street, Toad Town, SM, 12345 • Phone number (optional): blank • Delivery notes (optional): Please use doorbell. Click on “Place Order” button.
Expected Output	Pop an alert: “Shipping and billing addresses are different.” Alert includes two buttons: “Place Order” and “Go Back”.

Test Case ID	SYS12_HMTS_PopAddressAlert02
Purpose	Verify the system can detect inconsistency between billing and shipping address and prompt the user for action.
Test Setup	John has logged in into the system, added some products to his shopping cart, and requested to check out. He has selected “Pay by debit/credit card”.
Input	Fill out the fields required by this payment form: • Name on card: John Doe • Card number: 1234 5678 9123 4567 • Expiration date: 01/20 • Security code: blank • Billing address: 123 Main Street, Toad Town, SM, 12345 • Shipping address: 123 Main Street, Treetop Town, SM, 12345

	• Phone number (optional): blank • Delivery notes (optional): Please use doorbell. Click on “Place Order” button.
Expected Output	Pop an alert: “Shipping and billing addresses are different.” Alert includes two buttons: “Place Order” and “Go Back”.

Test Case ID	SYS12_HMTS_PopAddressAlert03
Purpose	Verify the system does not detect inconsistency between billing and shipping address when user chose to pay by cash.
Test Setup	John has logged in into the system, added some products to his shopping cart, and requested to check out. He has selected “Pay by cash”.
Input	Fill out the fields required by this payment form: • Shipping Address: 123 Main Street, Toad Town, SM, 12345 • Phone number (optional): blank • Delivery notes (optional): Please use doorbell. Click on “Place Order” button.
Expected Output	Since the user chose to pay by cash, address verification is not required. System will display order confirmation: • Shipping address: 123 Main Street, Toad Town, SM, 12345, • Payment method: cash • Delivery note: “Please use doorbell.”

7.4. Evaluation of Tests

In this section, we evaluate the tests by giving all the test results, and the code coverage for the 3 types of testing (unit test, subsystem test and system test).

7.4.1. Test Results

Here is the table showing the results of all the tests we perform. There are totally X tests which fail, and all of them are fixed properly.

Table 7.1. Test results for unit, subsystem and system tests.

Test Case ID	Pass/Fail	Fix
UNIT01_HMTS_CreateAccount01	Pass	N/A

UNIT01_HMTS_CreateAccount02	Pass	N/A
UNIT01_HMTS_CreateAccount03	Pass	N/A
UNIT01_HMTS_CreateAccount04	Pass	N/A
UNIT01_HMTS_CreateAccount05	Pass	N/A
UNIT02_HMTS_Login01	Pass	N/A
UNIT02_HMTS_Login02	Pass	N/A
UNIT02_HMTS_Login03	Pass	N/A
UNIT02_HMTS_Login04	Pass	N/A
UNIT02_HMTS_Login05	Pass	N/A
UNIT03_HMTS_Logout01	Pass	N/A
UNIT03_HMTS_Logout02	Pass	N/A
UNIT04_HMTS_RolesControl01	Pass	N/A
UNIT04_HMTS_RolesControl02	Pass	N/A
UNIT05_HMTS_VisualizeProducts01	Pass	N/A
UNIT05_HMTS_VisualizeProducts02	Pass	N/A
UNIT06_HMTS_AddProduct01	Pass	N/A
UNIT06_HMTS_AddProduct02	Pass	N/A
UNIT06_HMTS_AddProduct03	Pass	N/A
UNIT06_HMTS_AddProduct04	Pass	N/A
UNIT07_HMTS_VisualizeMenu01	Pass	N/A
UNIT07_HMTS_VisualizeMenu02	Pass	N/A
UNIT08_HMTS_SelectProduct04	Pass	N/A
UNIT08_HMTS_SelectProduct04	Pass	N/A
UNIT08_HMTS_SelectProduct04	Pass	N/A
UNIT08_HMTS_SelectProduct04	Pass	N/A

UNIT09_HMTS_ViewCart01	Pass	N/A
UNIT09_HMTS_ViewCart02	Pass	N/A
UNIT09_HMTS_ViewCart03	Pass	N/A
UNIT09_HMTS_ViewCart04	Pass	N/A
UNIT10_HMTS_RemoveProduct01	Pass	N/A
UNIT10_HMTS_RemoveProduct02	Pass	N/A
UNIT11_HMTS_FillOutOrderInfo01	Pass	N/A
UNIT11_HMTS_FillOutOrderInfo02	Pass	N/A
UNIT11_HMTS_FillOutOrderInfo03	Pass	N/A
UNIT11_HMTS_FillOutOrderInfo04	Pass	N/A
UNIT11_HMTS_FillOutOrderInfo05	Pass	N/A
UNIT11_HMTS_FillOutOrderInfo06	Pass	N/A
UNIT11_HMTS_FillOutOrderInfo07	Pass	N/A
UNIT11_HMTS_FillOutOrderInfo08	Pass	N/A
SUBSYS01_HMTS_CreateAccount01	Pass	N/A
SUBSYS01_HMTS_CreateAccount02	Pass	N/A
SUBSYS01_HMTS_CreateAccount03	Pass	N/A
SUBSYS01_HMTS_CreateAccount04	Pass	N/A
SUBSYS01_HMTS_CreateAccount05	Pass	N/A
SUBSYS01_HMTS_CreateAccount06	Pass	N/A
SUBSYS02_HMTS_Login01	Pass	N/A
SUBSYS02_HMTS_Login02	Pass	N/A
SUBSYS02_HMTS_Login03	Pass	N/A
SUBSYS02_HMTS_Login04	Pass	N/A

SUBSYS02_HMTS_Login05	Pass	N/A
SUBSYS03_HMTS_Logout01	Pass	N/A
SUBSYS04_HMTS_VisualizeProducts01	Pass	N/A
SUBSYS04_HMTS_VisualizeProducts02	Pass	N/A
SUBSYS05_HMTS_AddProduct01	Pass	N/A
SUBSYS05_HMTS_AddProduct02	Pass	N/A
SUBSYS05_HMTS_AddProduct03	Pass	N/A
SUBSYS06_HMTS_VisualizeMenu01	Pass	N/A
SUBSYS06_HMTS_VisualizeMenu02	Pass	N/A
SUBSYS07_HMTS_SelectProduct01	Pass	N/A
SUBSYS07_HMTS_SelectProduct02	Pass	N/A
SUBSYS07_HMTS_SelectProduct03	Pass	N/A
SUBSYS07_HMTS_SelectProduct04	Pass	N/A
SUBSYS08_HMTS_ViewCart01	Pass	N/A
SUBSYS08_HMTS_ViewCart02	Pass	N/A
SUBSYS08_HMTS_ViewCart03	Pass	N/A
SUBSYS09_HMTS_RemoveProduct01	Pass	N/A
SUBSYS09_HMTS_RemoveProduct02	Pass	N/A
SUBSYS10_HMTS_FillOutOrderInfo01	Pass	N/A
SUBSYS10_HMTS_FillOutOrderInfo02	Pass	N/A
SYS01_HMTS_CreateAccount01	Pass	N/A
SYS01_HMTS_CreateAccount02	Pass	N/A
SYS01_HMTS_CreateAccount03	Pass	N/A
SYS02_HMTS_Login01	Pass	N/A

SYS02_HMTS_Login02	Pass	N/A
SYS02_HMTS_Login03	Pass	N/A
SYS03_HMTS_Logout01	Pass	N/A
SYS03_HMTS_Logout02	Pass	N/A
SYS03_HMTS_Logout03	Pass	N/A
SYS04_HMTS_RolesControl01	Pass	N/A
SYS04_HMTS_RolesControl02	Pass	N/A
SYS04_HMTS_RolesControl03	Pass	N/A
SYS05_HMTS_VisualizeProducts01	Pass	N/A
SYS05_HMTS_VisualizeProducts02	Pass	N/A
SYS05_HMTS_VisualizeProducts03	Pass	N/A
SYS06_HMTS_AddProduct01	Pass	N/A
SYS06_HMTS_AddProduct02	Pass	N/A
SYS06_HMTS_AddProduct03	Pass	N/A
SYS07_HMTS_VisualizeMenu01	Pass	N/A
SYS07_HMTS_VisualizeMenu02	Pass	N/A
SYS07_HMTS_VisualizeMenu03	Pass	N/A
SYS08_HMTS_SelectProduct01	Pass	N/A
SYS08_HMTS_SelectProduct02	Pass	N/A
SYS08_HMTS_SelectProduct03	Pass	N/A
SYS09_HMTS_ViewCart01	Pass	N/A
SYS09_HMTS_ViewCart02	Pass	N/A
SYS09_HMTS_ViewCart03	Pass	N/A
SYS10_HMTS_RemoveProduct01	Pass	N/A
SYS10_HMTS_RemoveProduct02	Pass	N/A

SYS11_HMTS_FillOutOrderInfo01	Pass	N/A
SYS11_HMTS_FillOutOrderInfo02	Pass	N/A
SYS11_HMTS_FillOutOrderInfo03	Pass	N/A
SYS12_HMTS_PopAddressAlert01	Pass	N/A
SYS12_HMTS_PopAddressAlert02	Pass	N/A
SYS12_HMTS_PopAddressAlert03	Pass	N/A

7.4.2. Code Coverage

Here is the table showing the code coverage for the testing of our system, which includes unit testing, subsystem testing and systems testing.

Table 7.2. Code coverage for unit, subsystem and system tests.

Type of Testing	Number of Test Cases	Number of Bugs	Code Coverage
Unit	40	0	100% statement and branch coverage.
Subsystem	30	0	90% statement coverage of application logic subsystem. 85% branch coverage of application logic subsystem.
System	36	0	100% statement coverage of all implemented use cases.

8. Glossary

The following glossary is meant to ensure understanding of domain-specific terminology as it appears in the context of this document:

- **Actor**: Any individual or external entity interacting with the HMTS.
- **HMTS**: Happy Meal Take-out System.
- **Delivery man**: Restaurant employee (male or female) that is tasked with delivering products to customers. Will interact with the system mainly through the mobile application.
- **Manager**: Any restaurant employee with managerial privileges; manager can manage employee accounts in the system, alter menu, etc. They will interact with the system exclusively through the web page.
- **Staff**: The staff designation refers to any employee of a restaurant who does not fall into the Deliveryman or Manager categories. Will interact with the system exclusively through the web page.
- **Customer**: An individual who wishes to purchase food from a restaurant where the system is implemented.
- **Product**: Any food, side, or drink item available for sale by the restaurant; separated by Types, but all visualized on the same menu.
- **Roles**: Specific user groupings based on functionality; Manager, Customer, Deliveryman, Staff; used to specify Actors according to their privileges in the system.
- **User**: Generic entity in storage that encompasses all user roles. Corresponds to a customer, manager, deliveryman, or staff member registered in the system.
- **User (Reg):** Actors that are allowed to create a user account.
- **User (No Reg):** Actors that are not allowed to create a user account.
- **Artifact:** Specification of a physical piece of information used or produced by a software development process, or by deployment and operation of the system.
- **Node:** Computational resource upon which UML artifacts may be deployed for execution.
- **Use Case:** Encapsulation of various scenarios into one generic functionality requirement.

- **Non-functional requirements:** Requirements that support functional requirements in the software system. If the system does not meet non-functional requirements, its integrity is jeopardized.
- **Functional Requirements:** The functionality requested by the client of the software system. Includes security requirements.
- **Implementation:** The process of generating the functionalities put forth by the requirements.
- **Performance:** Non-functional requirement that details quantifiable performance data for a functionality of the system, usually in terms of the time it takes to complete the transaction.
- **Reliability:** Non-functional requirement that outlines how reliable a functionality should be, often expressed as Mean Time to Failure.
- **Supportability:** Non-functional requirement that details on what machines the software system can run on.
- **Usability:** Non-functional requirement that refers to how simple or difficult it is to activate a certain functionality or feature in a system.

9. Signature Page

The structure of the design document has been reviewed by all the team members. Our reviews were completed in accordance with the requests of the DD document.

Alex Marrero Pazos	Alex Marrero	04/27/2016
Name	Signature	Date

Jineth Quintero	Jineth Quintero	04/27/2016
Name	Signature	Date

Raul Camarca	Raul Camarca	04/27/2016
Name	Signature	Date

Weitong Liu	WEITONG LIU	04/27/2016
Name	Signature	Date

Wenbo Wang	Wenbo Wang	04/27/2016
Name	Signature	Date

10. References

The following sources have been quoted throughout this document:

[1] Zaikin, Mikalai. "Chapter 2. Common Architectures." Study Guide for Sun Certified Enterprise Architect for Java EE 5. Ixvar Inc, 22 Oct. 2007. Web. 10 Mar. 2016. <http://java.boot.by/scea5-guide/ch02s03.html>.

[2] Turnkey Linux. "MySQL." Turnkey Linux - MySQL. N.p., 15 Sept. 2015. Web. 12 Mar. 2016. <https://www.turnkeylinux.org/mysql>.

[3] Shaw, Mary, and David Garlan. Software Architecture: Perspectives on an Emerging Discipline. Upper Saddle River, NJ: Prentice Hall, 1996. Print. Vol. 1

[4] Object Management Group. "UML: Unified Modeling Language." SpringerReference 2nd ser. 2.1 (2007): 197. OMG. 4 Nov. 2007. Web. 10 Mar. 2016. <http://doc.omg.org/formal/2007-11-04.pdf>.

[5] Object Management Group. "UML: Unified Modeling Language." SpringerReference 2nd ser. 2.1 (2007): 210. OMG. 4 Nov. 2007. Web. 10 Mar. 2016. <http://doc.omg.org/formal/2007-11-04.pdf>.

11. Appendix

In this chapter we describe some additional information related to the whole project and system. It include project schedule, all use cases, user interfaces, detailed class diagrams, class interfaces, documented code for test driver and diary of meeting and tasks.

11.1. Appendix A – Project Schedule

In Figure 11.1.1 we describe schedule of project for phase 1 with a Gantt chart.

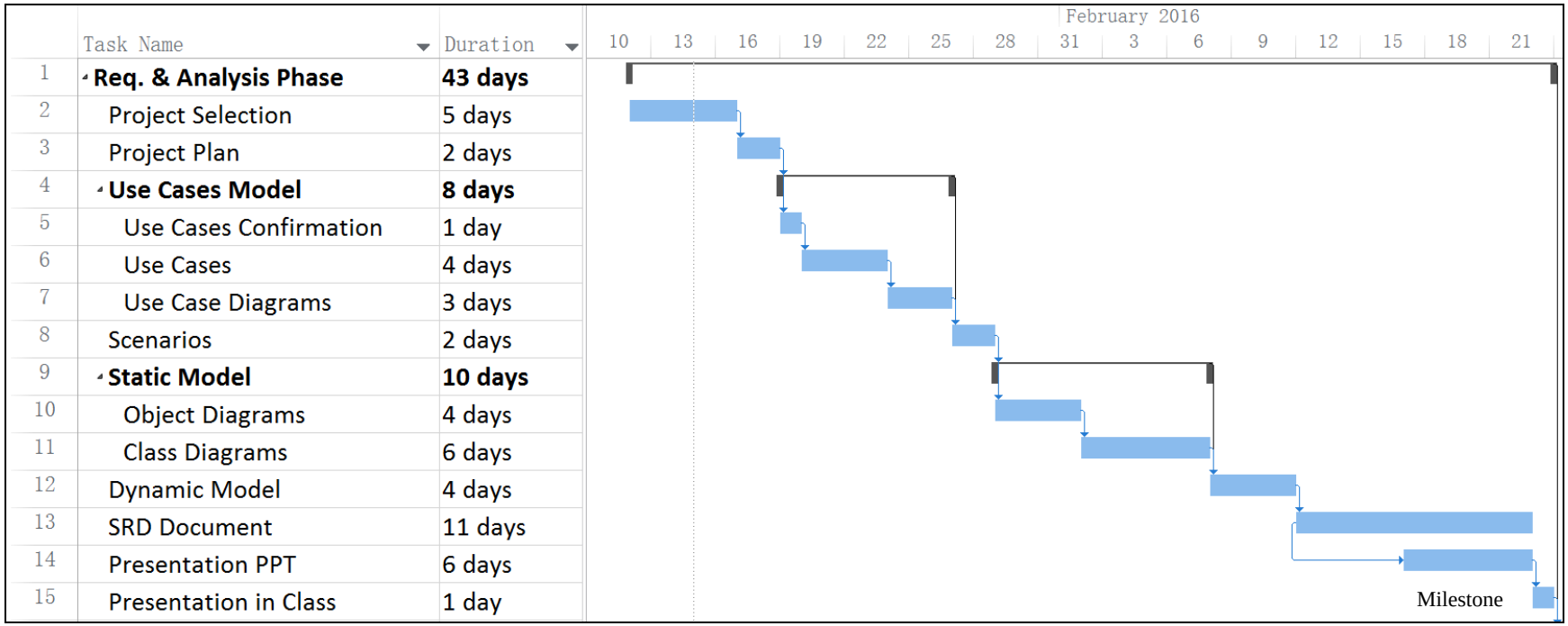
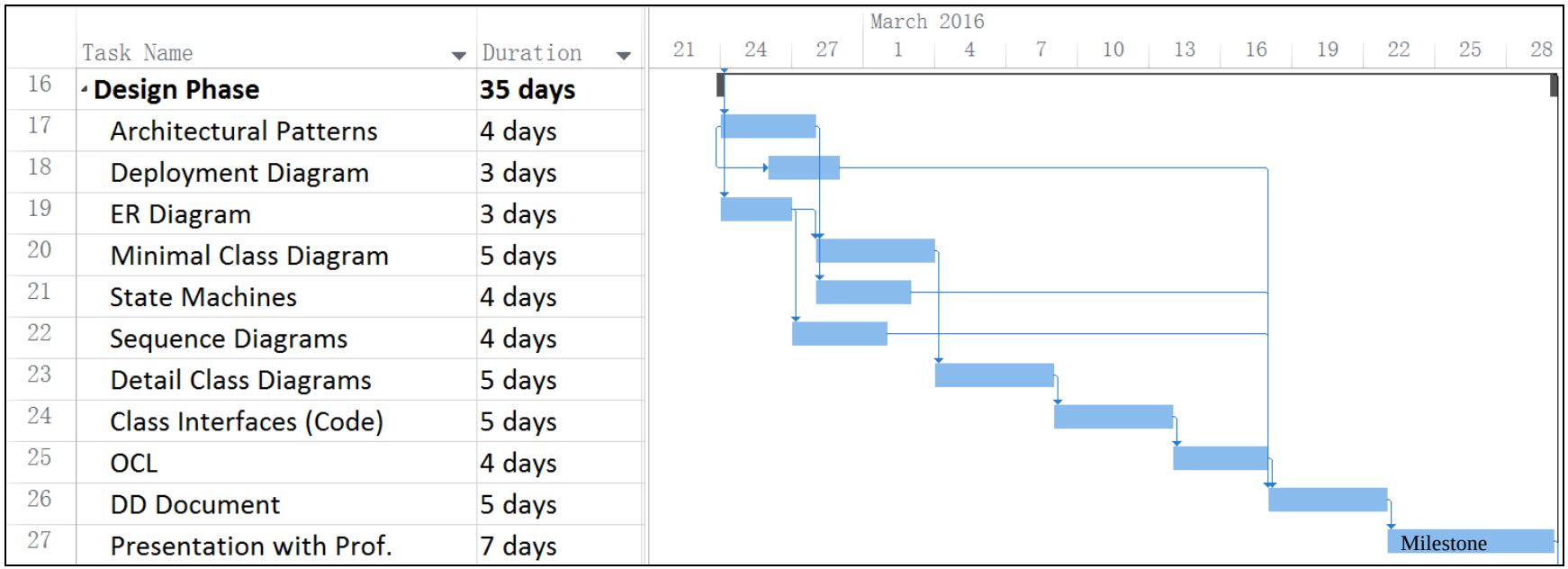


Figure 11.1.1. Schedule with Gantt chart for phase 1.

In Figure 11.1.2 we describe schedule of project for phase 2 with a Gantt chart.



	Task Name	Duration
16	**Design Phase**	**35 days**
17	Architectural Patterns	4 days
18	Deployment Diagram	3 days
19	ER Diagram	3 days
20	Minimal Class Diagram	5 days
21	State Machines	4 days
22	Sequence Diagrams	4 days
23	Detail Class Diagrams	5 days
24	Class Interfaces (Code)	5 days
25	OCL	4 days
26	DD Document	5 days
27	Presentation with Prof.	7 days

Figure 11.1.2. Schedule with Gantt chart for phase 2.

In Figure 11.1.3 we describe schedule of project for phase 3 with a Gantt chart.

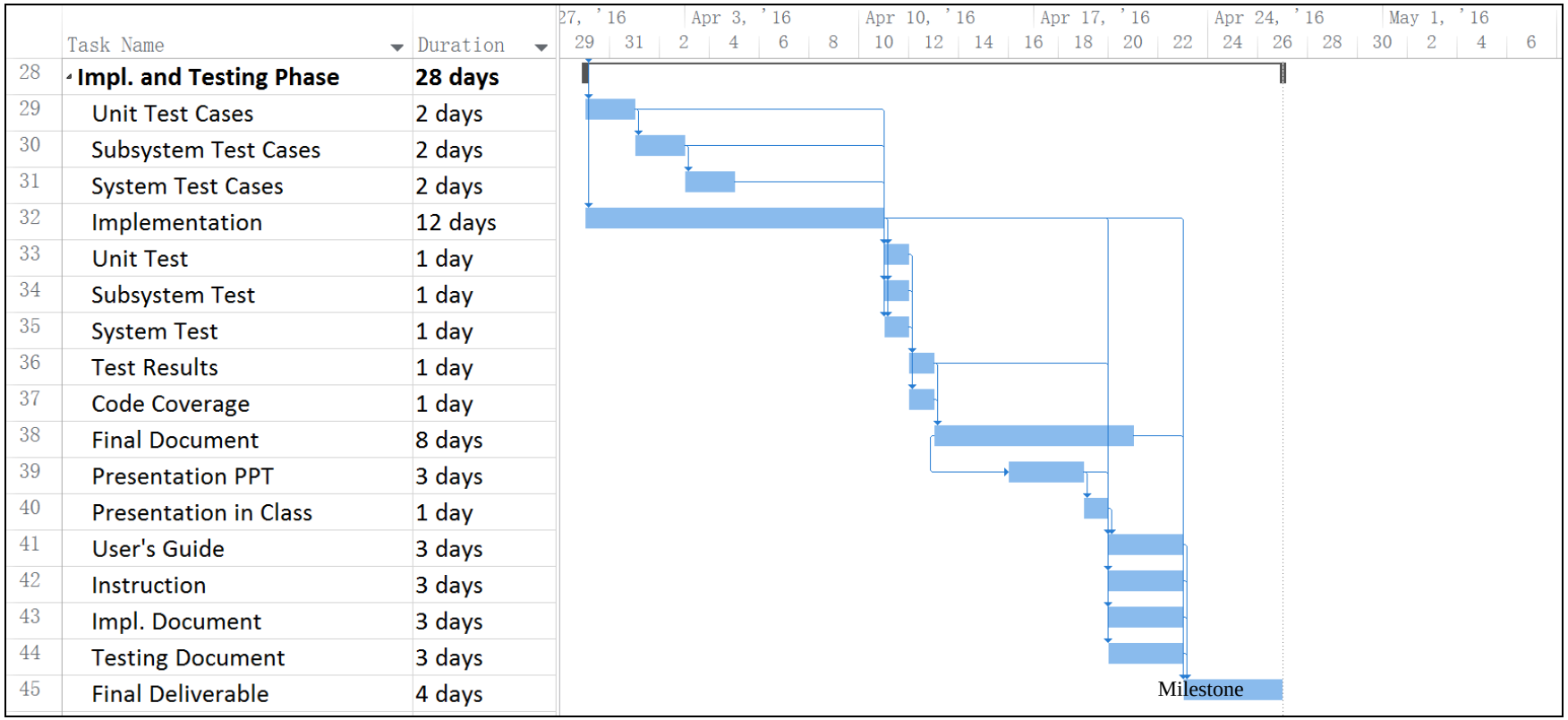


Figure 11.1.3. Schedule with Gantt chart for phase 3.

11.2. Appendix B – Use Cases with Nonfunctional Requirements

This section includes all the implemented use cases that incorporate non-functional requirements.

1. Create Account for Manager & Customer

Use case ID: HMTS_UM01_CreateAccount

Use Case level: High-level

Details:

- **Actor:** Manager, Customer
- **Pre-conditions:** Customer is on main login screen; creating account option is available; customer is not already in system.
- **Description:** Allow customer to create login credentials. Management will create management account and accounts for operational staff and deliverymen.
- **Trigger:** The actor selects the creating account option.

  The system responds by:

  1. Showing a page with empty key fields that the actor can add his/her information to. Key fields include:
     a. First name
     b. Last name
     c. Address
     d. Phone number
     e. Username/Email
     f. Password
     g. Security Question/Answer
  2. If management is creating account, system will prompt for additional information about the type of actor: Customer; Management; Staff; Deliveryman.
  3. Actor will click creating account button.

     If successful, an account is created.

     If the actor account can already exist (verified through email). The system will let the actor know and prompt for another email.

- **Relevant Requirements:** None
- **Post-Conditions:** Customer account is in the system; the customer is ready to order

Alternate Courses of Action: None

Extensions: None

Exceptions: None

Concurrent Uses: None

Related Use Cases: HMTS_UM02_Login, HMTS_UM03_ForgetPassword

Decision Support:

Frequency: Medium – Function will be accessed 5 times per hour.

Criticality: Medium – Actors can still use system for its intended functionality without this function.

Risk: Low – This function is explicit and can be implemented accurately.

Constraints:

- **Usability**

  No previous training is needed.

- **Reliability**

  Mean time to failure – 1 failure for every 24 hours of operation.

- **Performance**

  The procedure should take no more than 2 seconds.

- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system is to be done in Java in the Eclipse development environment.

Modification History

- **Owner:** Alex Marrero Pazos
- **Initiation Date:** 01/21/2016
- **Last Modified:** 03/23/2016

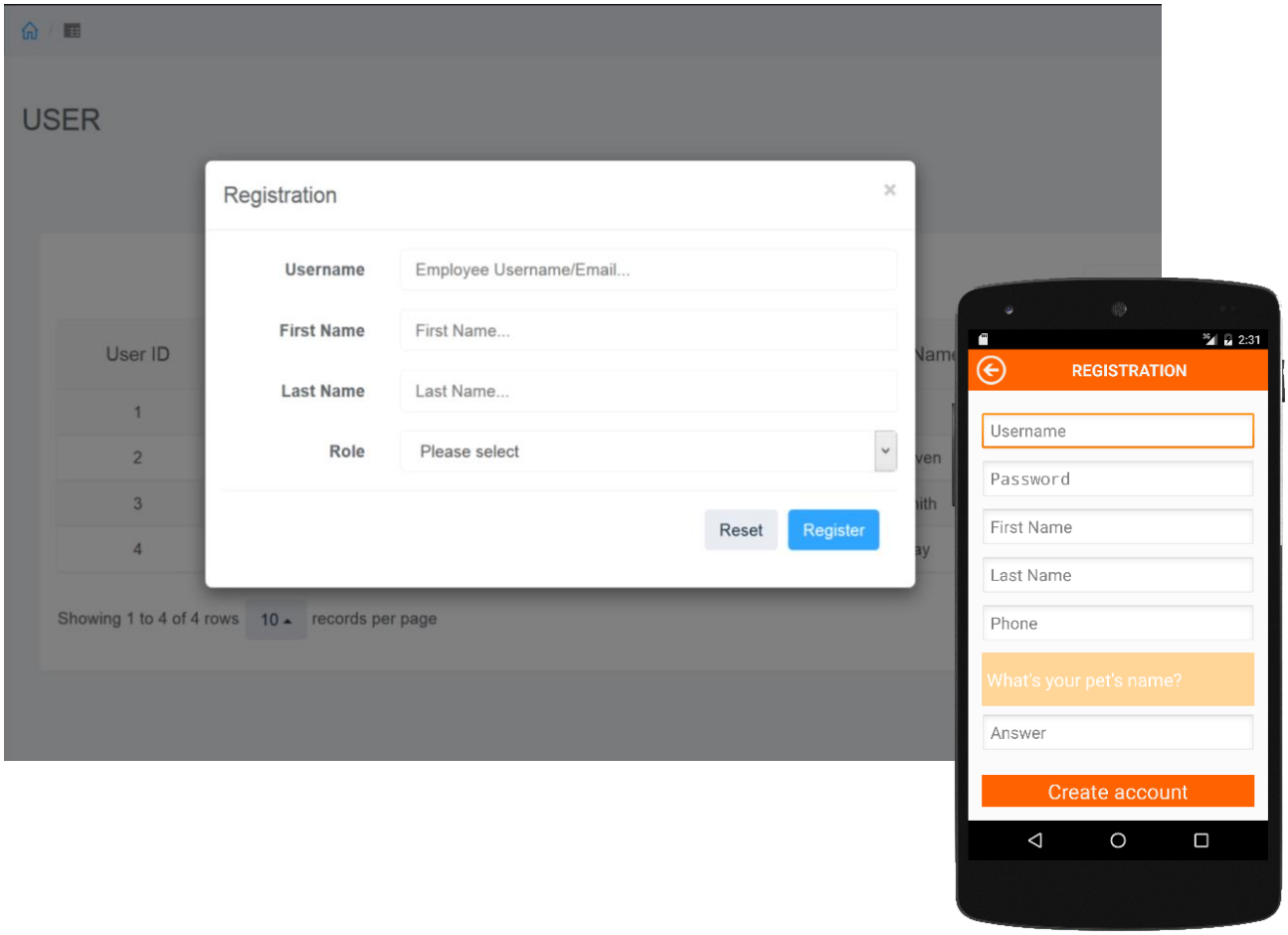


Figure 11.2.1. Registration page for mobile and web.

2. *Login System for User*

Use Case ID: HTMS_UM02_Login

Use Case Level: High-Level

Details:

- **Actor:** Customer, Manager, Staff, Delivery man
- **Pre-conditions:** The actor has a valid account and entered the login page of the system.
- **Description:** The actor will login into the system to be able to use the functionlaity available according with its role.
- **Trigger:** The actor type its credentials and clicks the button “Login”.
  The system responds by

1. The system verify the user name exist.
2. The system verify the password corresponds to the username.
3. The system validates the information:
    a. If the credentials are correct the actor is logged into the system.
    b. If the credentials are incorrect the system displays a message letting know the actor.
4. The system shows the functionality according to the role of the actor.

- **Relevant requirements:** None
- **Post-conditions:** The actor is logged into the system and is able to use the functionality according to its role.

Alternative Courses of Action: HTMS_UM03_ForgetPassword

Extensions: HTMS_UM03_ForgetPassword, HTMS_UM05_LoginTimeout, HTMS_UM06_LimitFailedLoginTimes

Exceptions: None

Concurrent Uses: None

Related Use Cases: HTMS_UM01_CreateAccount

Decision Support

Frequency: High – the actors can login into the system anytime they they want to use the functionality (100 times per hour).

Criticality: High – the actor cannot use the functionality of the system without been logged.

Risk: Low – This function is explicit and can be implemented accurately.

Constraints:

- **Usability**

  No previous traininng is needed.
- **Reliability**

  Mean Time to Failure – 1 failure for every 24 hours of operation is acceptable.
- **Performance**

  The login should take no more than 2 seconds.
- **Supportability**

The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed..

- **Implementation**

  The implementation of the system will be done in Java and Eclipse framework and Android framework.

Modification History

Owner: Jineth Quintero

Initiation date: 01/19/2016

Date last modified: 03/23/2016

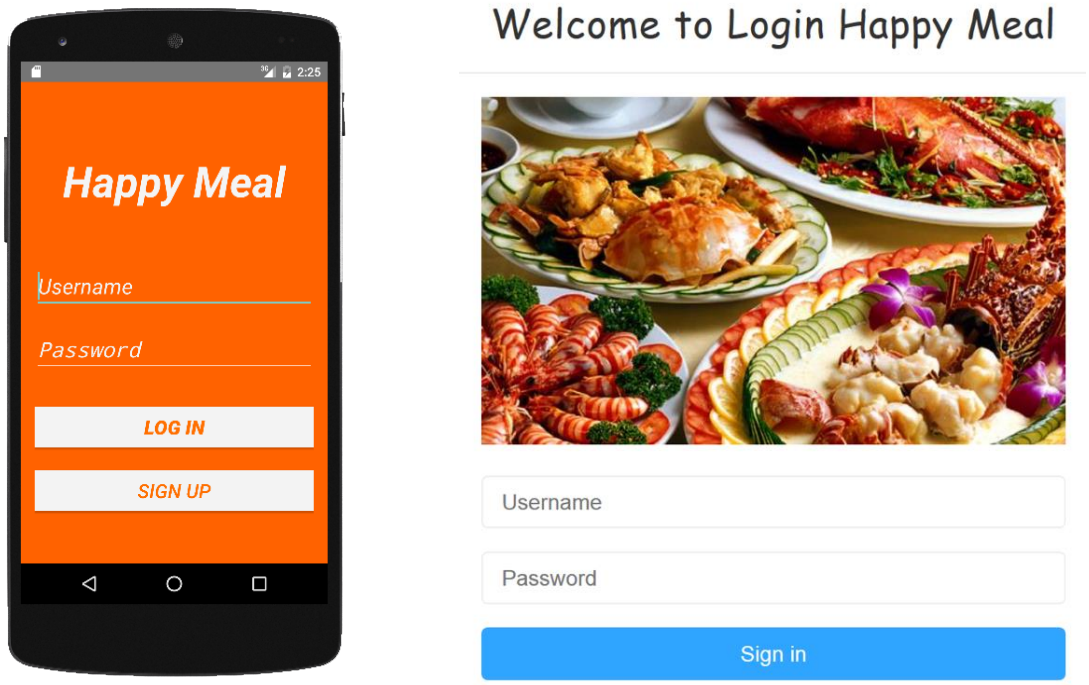


Figure 11.2.2. Login page for mobile and web.

3. *Forget Password for User - Security*

Use Case ID: HTMS_UM03_ForgetPassword

Use Case Level: System-Level End-to-End

Details:

- **Actor:** Customer, Manager, Staff, Delivery man
- **Pre-conditions:** The actor has a valid account and entered the login page of the system.
- **Description:** The actor will be able to ask for a reminder in case it forgot the password.
- **Trigger:** The actor clicks the button “Forgot password”.

  The system responds by

  1. The system ask the actor for the username.
  2. The actor type his username and clicks the button “Continue”.
  3. The system verify if the username exist.
     a. If the username doesn’t exist the system displays a message notifying the actor.
     b. If the username exist, the system asks the actor for the answer of the security question.
  4. The actor types the answer of the security question and clicks the button “Get password”.
  5. The system verify if the answer is correct.
     a. If the answer is correct the system displays a message showing the user’s password.
     b. If the answer is not correct the system displays a message notifying the user.
- **Relevant requirements:** None
- **Post-conditions:** The actor will see its password.

Alternative Courses of Action None

Extensions: None

Exceptions: None

Concurrent Uses: None

Related Use Cases: HMTS_UM02_Login

Decision Support

Frequency: Medium – the actors will use the reminder for the password everytime they forget the password (20 times per hour).

Criticality: High – the actor cannot use the functionality of the system without been logged, that makes important to be able to obtain a remainder of the password in case is been forgotten.

Risk: Low – This function is explicit and can be implemented accurately.

Constraints:

- **Usability**

  No previous training is needed.

- **Reliability**

  Mean Time to Failure – 1 failure for every 24 hours of operation is acceptable.

- **Performance**

  The process should take no more than 2 seconds

- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system will be done in Java and Eclipse framework and Android framework.

Modification History

Owner: Jineth Quintero

Initiation date: 01/21/2016

Date last modified: 03/23/2016

4. *Logout System for User*

Use Case ID: HTMS_UM04_Logout

Use Case Level: High-Level

Details:

- **Actor:** Customer, Manager, Staff, Delivery man
- **Pre-conditions:** The actor has a valid account and is logged into the system.
- **Description:** The actor will be able to logout.
- **Trigger:** The actor clicks the button "logout"

  The system responds by

  1. The system logs out the actor.
  2. The system displays login page to indicate the actor it has been successfully logged out.
- **Relevant requirements:** None
- **Post-conditions:** The actor is logged out and have no access to the functionality of the system.

Alternative Courses of Action: None

Extensions: None

Exceptions: None

Concurrent Uses: None

Related Use Cases: HTMS_UM02_Login

Decision Support

Frequency: High – Performed everytime the actor want to sign out the system (100 times per hour).

Criticality: Low – The use of the functionality of the system is not going to be affected if the user didn't logout.

Risk: Low – This function is explicit and can be implemented accurately.

Constraints:

- **Usability**

  No previous training is needed.
- **Reliability**

Mean Time to Failure – 1 failure for every 24 hours of operation is acceptable.

- **Performance**

  The logout should take no more than 2 seconds

- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed..

- **Implementation**

  The implementation of the system will be done in Java and Eclipse framework and Android framework.

Modification History

Owner: Jineth Quintero

Initiation date: 01/21/2016

Date last modified: 03/23/2016

---

5. ***Login Timeout for Users - Security***

Use case ID: HMTS_UM05_LoginTimeout

Use Case level: System level end-to-end

Details:

- **Actor:** Manager, Staff
- **Pre-conditions:** The actor's account is logged in to system.
- **Description:** The actor abandons device while logged into system for longer than 5 minutes. The account will be logged out itself due to inactivity.
- **Trigger:** The actor leaves computer unattended.

  The system responds by

  1. Resetting a timer every time the actor does something in the system.
  2. After the timer exceeds 5 minutes, the system will log the actor out.
- **Relevant Requirements:** None
- **Post-Conditions:** The actor account is logged out.

Alternate Courses of Action: None

Extensions: None

Exceptions: None

Concurrent Uses: None

Related Use Cases: HMTS_UM02_Login

Decision Support:

Frequency: Medium – Most legitimate users will not leave their devices unattended; 10 times per day.

Criticality: Medium – Actors can still use system for its intended functionality without this function.

Risk: Low – This function is explicit and can be implemented accurately.

Constraints:

- **Usability**

  No previous training is needed.

- **Reliability**

  Mean time to failure – 1 failure for every 24 hours of operation.

- **Performance**

  The procedure should take no more than 2 seconds.

- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system is to be done in Java in the Eclipse development environment.

Modification History

- **Owner:** Alex Marrero Pazos
- **Initiation Date:** 01/21/2016
- **Last Modified:** 04/08/2016

6. *Limit Failed Login Times for Users - Security*

Use Case ID: HMTS_UM06_LimitFailedLoginTimes

Use Case Level: System-Level End-to-End

Details:

- **Actor:** Customer, Manager, Employee and Delivery man
- **Pre-conditions:** The system is running. The actor has a username.
- **Description:** The actor is allowed to attempt to login for only 5 times. After that, the user's account will be frozen for 5 minutes.
- **Trigger:** The actor initiates an action by typing his username and password, then clicking the "login" button on a login page of the system.

  The system responds by

  1. The system receives the request and verifies whether the actor's credential is valid.
     a. If the credentials are correct, then the actor logs into the system and is allowed to visualize corresponding functions.
     b. Else, the system displays a message "Invalid username or password", and allows the actor to try again. After the actor has attempted for 5 times and still failed, his account will be frozen for 5 minutes by the system.
- **Relevant requirements:** None
- **Post-conditions:** if the actor's account is frozen, he may use "Forget password" to get his password back and try to login again.

Alternative Courses of Action: None

Extensions: None

Exceptions: None

Concurrent Uses: None

Related Use Cases: None

Decision Support

Frequency: High – Performed for every time actors want to login the system. (100 times per hour)

Criticality: High – It's an important function for the safety of users' accounts.

Risk: Low – This function is explicit and can be implemented accurately.

Constraints:

- **Usability**

  No previous training is needed.

- **Reliability**

  Mean Time to Failure – 1 failure for every 24 hours of operation is acceptable.

- **Performance**

  The login will be processed in 2 seconds.

- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system is to be done in Java, the Eclipse framework and Android framework.

Modification History

Owner: Weitong Liu

Initiation date: 01/22/2016

Date last modified: 04/08/2016

---

7. Roles Control for User - Security

Use Case ID: HMTS_UM07_RolesControl

Use Case Level: System-Level End-to-End

Details:

- **Actor:** Customer, Manager, Staff and Delivery man
- **Pre-conditions:** The system is running. The actor is signing in the system.
- **Description:** The system shall use roles control for different roles of actors. Different roles have different authorities to the functions of the system.

- **Trigger: :** The actor initiates an action by inputting credential and the sysem verifies the credential is valid.

  The system responds by

  1. The system visualizes the corresponding functions according to the role of the actor.
- **Relevant requirements:** None
- **Post-conditions:** An actor can use the corresponding functions according to the role of the actor.

Alternative Courses of Action: None

Extensions: None

Exceptions: If the system can not decide which role an actor belongs to, the system will throw an exception.

Concurrent Uses: None

Related Use Cases: None

Decision Support

Frequency: High – Performed for every time actors want to login the system. (100 times per hour)

Criticality: High – It's a important function for actors to access their corresponding functions and to protect the security of the system.

Risk: Low – This function is explicit and can be implemented accurately.

Constraints:

- **Usability**

  No previous training is needed.
- **Reliability**

  Mean Time to Failure – 1 failure for every 24 hours of operation is acceptable.
- **Performance**

  The procedure of roles decision will be completed within 2 seconds.
- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system is to be done in Java, the Eclipse framework and Android framework.

Modification History

Owner: Wenbo Wang

Initiation date: 01/21/2016

Date last modified: 03/23/2016

---

8. Visualize Products for Manager & Staff

Use Case ID: HMTS_OP01_VisualizeProducts

Use Case Level: High-Level

Details:

- **Actor:** Manager, Staff
- **Pre-conditions:** The actor has signed in the system.
- **Description:** The system shall visualize a product-list containing dishes, sides and drinks for the actor. For each product, it contains the name, type and price.
- **Trigger:** The actor initiates an action by clicking "Prodcut" menu item.

  The system responds by

  1. The system displays all the products information into a list to the actor. If there is not any product, it will only show the titles of each column in the list.
- **Relevant requirements:** None
- **Post-conditions:** The system shows the information of all products. The Actor can know what products the restaurant has now and may add and remove products.

Alternative Courses of Action: None

Extensions: HMTS_OP02_AddProduct, HMTS_OP03_RemoveProduct

Exceptions: None

Concurrent Uses: None

Related Use Cases: HMTS_OP02_AddProduct, HMTS_OP03_RemoveProduct

---

Decision Support

Frequency: Medium – Performed for every time actors want to view or edit products. (20 times per day)

Criticality: High – It's a useful function for products management.

Risk: Low – This function is explicit and can be implemented accurately.

Constraints:

- **Usability**

  No previous training is needed.

- **Reliability**

  Mean Time to Failure – 1 failure for every 24 hours of operation is acceptable.

- **Performance**

  The products information will be shown within 2 seconds.

- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system is to be done in Java and the Eclipse framework.

Modification History

Owner: Wenbo Wang

Initiation date: 01/20/2016

Date last modified: 03/23/2016

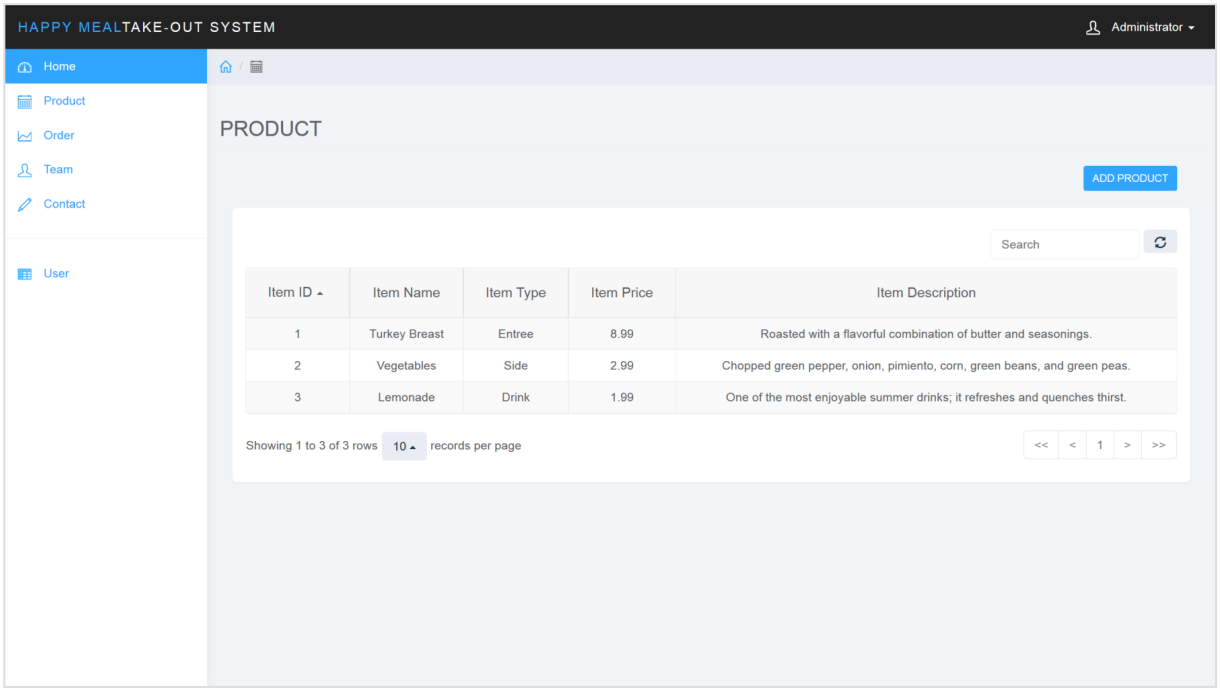


Figure 11.2.3. Visualize products page for Manager and Staff.

9. *Add Product for Manager & Staff*

Use Case ID: HMTS_OP02_AddProduct

Use Case Level: High-Level

Details:

- **Actor:** Manager, Staff
- **Pre-conditions:** The actor has signed in the system and entered the "Product" web page.
- **Description:** The system allow the actor to add dishes, sides or drinks to a menu. For each product, it contains the name, type, price and optional brief.
- **Trigger:** The actor initiates an action by pushing "Add" button on the "Product" page.

  The system responds by

  1. The system shows a web page for adding a product.
  2. The actor fill out the information of a new product into the key fields including name, type, price and optional brief.
  3. If the actor pushes the "Submit" button:

a. If the actor misses any necessary info and push “Submit” button, the system will show an prompt that indicates which info the actor misses.
b. If the actor inputs all the necessary info, it will go to step 4.
c. If the actor pushes “Cancel” button to close the page, and the system will go back to the products-list web page.

4. The system saves the product into the menu.

- **Relevant requirements:** None
- **Post-conditions:** The new product is added into the menu; it can be checked on the products-list.

Alternative Courses of Action: None

Extensions: None

Exceptions: None

Concurrent Uses: None

Related Use Cases: HMTS_OP01_VisualizeProducts

Decision Support

Frequency: Low – Performed for every time actors want to add a product. (20 times per month)

Criticality: High – It’s a important function to add a new product into a menu.

Risk: Low – This function is explicit and can be implemented accurately.

Constraints:

- **Usability**

  No previous training is needed.
- **Reliability**

  Mean Time to Failure – 1 failure for every 24 hours of operation is acceptable.
- **Performance**

  The product will be added into a menu within 2 seconds.
- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.
- **Implementation**

The implementation of the system is to be done in Java and the Eclipse framework.

Modification History

Owner: Wenbo Wang

Initiation date: 01/20/2016

Date last modified: 03/23/2016

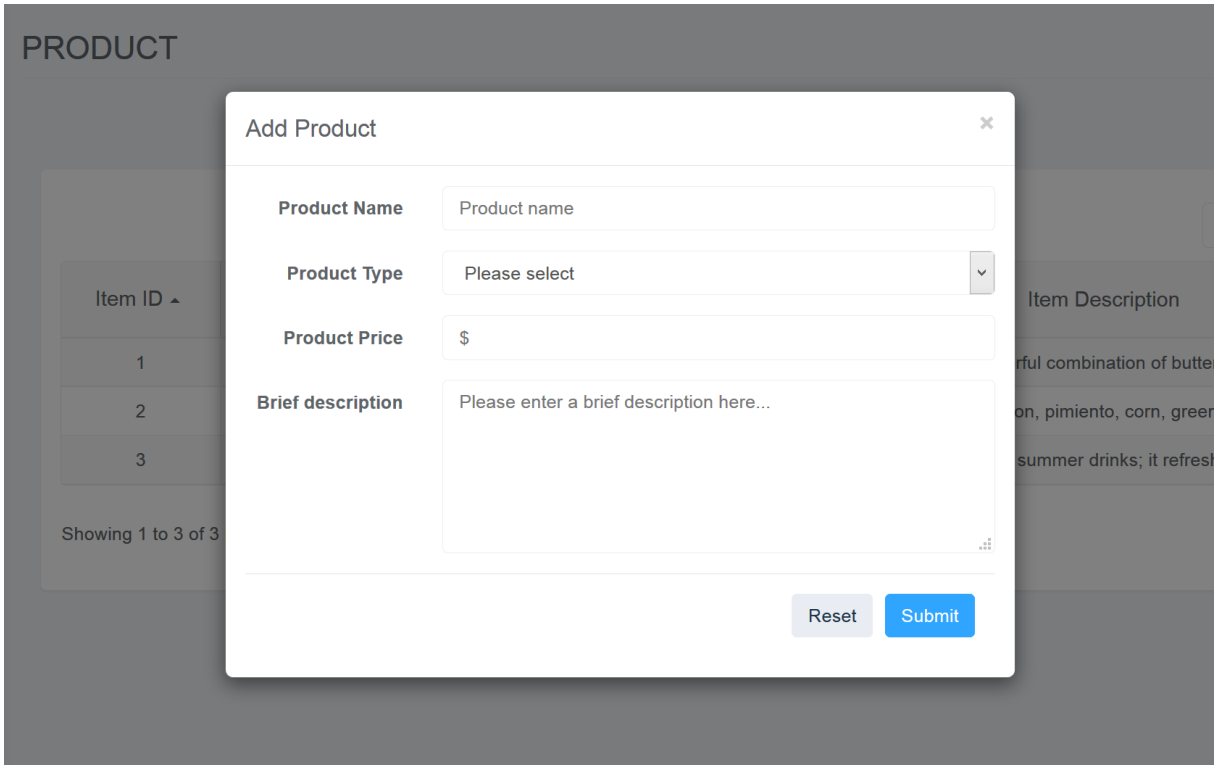


Figure 11.2.4. Add product page for Manager and Staff.

10. Remove Product for Manager & Staff

Use case ID: HMTS_OP03_RemoveProduct

Use Case level: High-Level

Details:

- **Actor:** Manager, Staff
- **Pre-conditions:** The actor is logged in. An item is currently available for purchase.

- **Description:** Allow the actor to alter the menu by deleting a certain item from the menu.
- **Trigger:** The actor will click on the "Delete" button adjacent a certain item. Alternatively, the actor will go into that particular item's menu of options and delete from there.

  The system responds by:

  1. Deleting the item from the menu. If item is currently in a customer's shopping cart, it will prompt before deletion.
- **Relevant Requirements:** None
- **Post-Conditions:** Item is deleted from the system.

Alternate Courses of Action: None.

Extensions: None

Exceptions: None

Concurrent Uses: None

Related Use Cases: HMTS_OP02_AddProduct

Decision Support:

Frequency: Low – Function will be accessed once per day.

Criticality: Medium – Actors can still use system for its intended functionality without this function.

Risk: Low – This function is explicit and can be implemented accurately.

Constraints:

- **Usability**

  No previous training is needed.
- **Reliability**

  Mean time to failure – 1 failure for every 24 hours of operation.
- **Performance**

  The procedure should take no more than 2 seconds.
- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system is to be done in Java in the Eclipse development environment.

Modification History

- **Owner:** Alex Marrero Pazos
- **Initiation Date:** 01/21/2016
- **Last Modified:** 04/08/2016

---

11. Visualize Orders for Manager & Staff

Use Case ID: HTMS_OP04_VisualizeOrders

Use Case Level: High-Level

Details:

- **Actor:** Manager, Staff
- **Pre-conditions:** The actor has a valid account and has signed in the system and enter the "Order" web page.
- **Description:** The actor will be able to view a list of orders.
- **Trigger:** The actor select the option view orders.

  The system responds by

  1. The system displays a list of orders. If there are no orders, the system will display a message letting the user know it.
- **Relevant requirements:** None
- **Post-conditions:** The actor will be able to see a list of orders.

Alternative Courses of Action None

Extensions: HTMS_OP05_VisualizeDetailedOrder

Exceptions: None

Concurrent Uses: None

Related Use Cases: HTMS_OP05_VisualizeDetailedOrder

Decision Support

Frequency: High – the users will obtain a list of pending orders (50 times per hour).

Criticality: High – the user can not process an order without having the necesry information.

Risk: Low – This function is explicit and can be implemented accurately.

Constraints:

- **Usability**

  No previous training is needed.

- **Reliability**

  Mean Time to Failure – 1 failure for every 24 hours of operation is acceptable.

- **Performance**

  The list of orders should take no more than 2 seconds

- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed..

- **Implementation**

  The implementation of the system will be done in Java and Eclipse framework.

Modification History

Owner: Jineth Quintero

Initiation date: 01/21/2016

Date last modified: 04/08/2016

---

12. Visualize DetailedOrder for Manager & Staff

Use Case ID: HTMS_OP05_VisualizeDetailedOrder

Use Case Level: System-Level End-to-End

Details:

- **Actor:** Manager, Staff

- **Pre-conditions:** The actor has a valid account and has signed in the system and entered the "Order" web page containing a list of orders.
- **Description:** The actor will be able to view a detailed information of an order.
- **Trigger:** The actor select the option view detail of order.

  The system responds by

  1. The system displays a detailed information of the selected order.
- **Relevant requirements:** None
- **Post-conditions:** The actor will be able to see a detailed information of an order.

Alternative Courses of Action None

Extensions: None

Exceptions: None

Concurrent Uses: None

Related Use Cases: HTMS_OP04_VisualizeOrder

Decision Support

Frequency: High – the users will obtain a detailed information of a specific order (100 times per hour).

Criticality: High – the visualize detailed order is necessary to be able to process the orders for each customer.

Risk: Low – This function is explicit and can be implemented accurately.

Constraints:

- **Usability**

  No previous training is needed.
- **Reliability**

  Mean Time to Failure – 1 failure for every 24 hours of operation is acceptable.
- **Performance**

  The process should take no more than 2 seconds
- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed..
- **Implementation**

The implementation of the system will be done in Java and Eclipse framework.

Modification History

Owner: Jineth Quintero

Initiation date: 01/21/2016

Date last modified: 04/08/2016

---

13. Process Order for Manager & Staff

Use case ID: HMTS_OP06_ProcessOrder

Use Case level: High-Level

Details:

- **Actor:** Manager, Staff
- **Pre-conditions:** The actor is logged in. A particular order has been placed, paid for, and filled by the restaurant.
- **Description:** The actor will have the ability to process an order after the restaurant completes it. This function serves as the final step and transitions the order to the deliveryman user. This should happen concurrently to the restaurant handing over the order to the deliveryman.
- **Trigger:** The actor will click "Process Order"

  The system responds by

  1. Updating order status to "Filed".
  2. Sending order information to delivery man.
- **Relevant Requirements:** None
- **Post-Conditions:** Order is visible to delivery man. Delivery man should have physical possession of the order.

Alternate Courses of Action: None.

Extensions: None

Exceptions: None

Concurrent Uses: None

Related Use Cases: HMTS_OP05_VisualizeDetailedOrder, HMTS_CU08_PlaceOrder

---

Decision Support:

Frequency: High – Function will be accessed 100 times per hour.

Criticality: High – Critical system to intended application functionality.

Risk: Medium – This function is explicit, but the mobile technique used to implement the system is new for the development team.

Constraints:

- **Usability**

  No previous training is needed

- **Reliability**

  Mean time to failure – 1 failure for every 24 hours of operation.

- **Performance**

  The procedure should take no more than 2 seconds.

- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system is to be done in Java in the Eclipse development environment.

Modification History

- **Owner:** Alex Marrero Pazos
- **Initiation Date:** 01/21/2016
- **Last Modified:** 04/08/2016

14. Visualize Menu for Customer

Use Case ID: HMTS_CU01_VisualizeMenu

Use Case Level: High-Level

Details:

- **Actor:** Customer
- **Pre-conditions:** The actor has signed in the system.
- **Description:** The system shall visualize a menu including dishes, sides and drinks for the actor. For each product, it contains the name, type, price and optional brief.
- **Trigger:** The actor initiates an action by clicking "menu" menu item.

  The system responds by

  1. The system visualizes the menu. If there is no product, it will show blank on the menu.
- **Relevant requirements:** None
- **Post-conditions:** The system shows the menu of the restaurant. The Actor can know what products they may purchase.

Alternative Courses of Action: None

Extensions: HMTS_CU02_AddProduct, HMTS_CU03_RemoveProduct

Exceptions: None

Concurrent Uses: None

Related Use Cases: HMTS_CU02_AddProduct, HMTS_CU03_RemoveProduct

---

Decision Support

Frequency: High – Performed for every time the actor wants to view a menu or a cart, and add products to a cart. (200 times per hour)

Criticality: High – It's a important function for the actor to choose dishes, sides or drinks from a menu.

Risk: Medium – This function is explicit, but the mobile technique used to implement the system is new for the development team.

Constraints:

- **Usability**

  No previous training is needed.
- **Reliability**

Mean Time to Failure – 1 failure for every 24 hours of operation is acceptable.

- **Performance**

  The menu will be shown within 2 seconds.

- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system is to be done in Java, the Eclipse framework and Android framework.

Modification History

Owner: Wenbo Wang

Initiation date: 01/21/2016

Date last modified: 03/23/2016

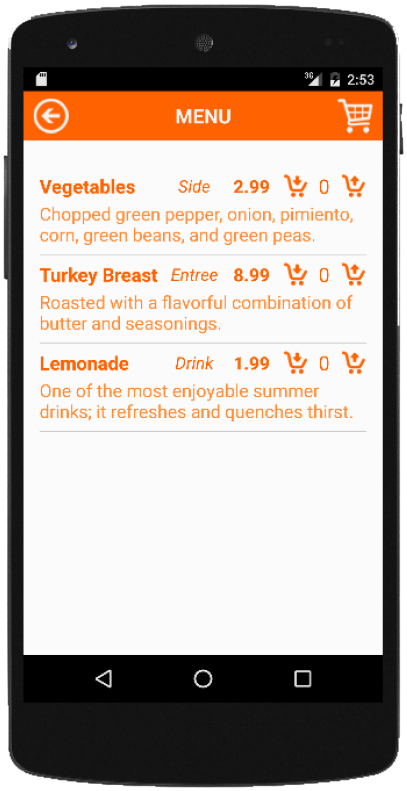


Figure 11.2.5. Visualize menu page for Customer.

15. Add Product for Customer

Use Case ID: HMTS_CU02_AddProduct

Use Case Level: High-Level

Details:

- **Actor:** Customer
- **Pre-conditions:** The actor has signed in the system and entered the "Menu" page.
- **Description:** The system allow the actor to add dishes, sides or drinks into a cart.
- **Trigger:** The actor initiates an action by starting inputting the number of a product the actor wants to add into a cart.

  The system responds by

  1. The actor inputs the number of a product that the actor wants to add into a cart.
  2. The actor pushes "Add" button for the product on the "Menu" page.
     a. if the number of the product the actor wants to purchase is less than 1 or not an integer, the system will pop up an alter.
     b. otherwise, the system will go to step 3.
  3. The system saves the product and its quantity into a cart.
- **Relevant requirements:** None
- **Post-conditions:** The record of the products and its quantity is saved into the cart; it can be checked in the cart.

Alternative Courses of Action: None

Extensions: None

Exceptions: None

Concurrent Uses: None

Related Use Cases: HMTS_CU01_VisualizeMenu

Decision Support

Frequency: High – Performed for every time actors want to add a product into their carts. (500 times per hour)

Criticality: High – It's a important function to add a product into a cart.

Risk: Medium – This function is explicit, but the mobile technique used to implement the system is new for the development team.

Constraints:

- **Usability**

  No previous training is needed.

- **Reliability**

  Mean Time to Failure – 1 failure for every 24 hours of operation is acceptable.

- **Performance**

  The product will be added to a cart in 1 seconds.

- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system is to be done in Java, the Eclipse framework and Android framework.

Modification History

Owner: Wenbo Wang

Initiation date: 01/21/2016

Date last modified: 03/23/2016

---

16. View Cart for Customer

Use Case ID: HMTS_CU03_ViewCart

Use Case Level: High-Level

Details:

- **Actor:** Customer
- **Pre-conditions:** The system is running. The actor has signed in the system.

- **Description:** The system shall visualize a cart containing the information about the products the actor has chosen. For each item, it contains the product name, price, number, and total price.
- **Trigger:** The actor initiates an action by clicking "View my cart" item on the navigation bar.

  The system responds by

  1. The system visualizes the corresponding products' information for the current actor. If there is not any product, it will only show the titles of each column in the cart.
- **Relevant requirements:** None
- **Post-conditions:** The actor can see all the products he has chosen now and performs some operations on the cart, such as change the number of a product, delete a product from the cart, view details of a product, and so on.

Alternative Courses of Action: None

Extensions: HMTS_CU04_RemoveProduct

Exceptions: None

Concurrent Uses: None

Related Use Cases: HMTS_CU02_AddProduct, HMTS_CU04_RemoveProduct

---

Decision Support

Frequency: Medium – Performed for every time a customer want to view his cart. (30 times per day)

Criticality: High – It's a useful function for a customer to manage the products he has chosen.

Risk: Low – This function is explicit and can be implemented accurately.

Constraints:

- **Usability**

  No previous training is needed.
- **Reliability**

  Mean Time to Failure – 1 failure for every 24 hours of operation is acceptable.
- **Performance**

  The information of products in the cart will be shown in 2 seconds.

- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system is to be done in Java, the Eclipse framework and Android framework.

Modification History

Owner: Weitong Liu

Initiation date: 01/25/2016

Date last modified: 03/23/2016

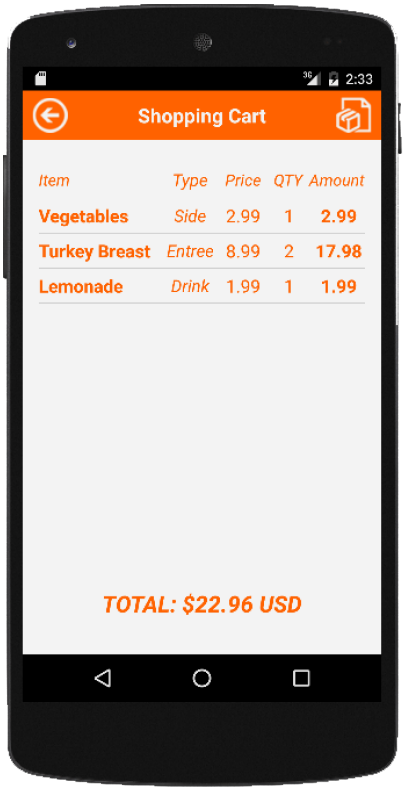


Figure 11.2.6. Visualize shopping cart page for Customer.

17. Remove Product for Customer

Use case ID: HMTS_CU04_RemoveProduct

Use Case level: High-Level

Details:

- **Actor:** Customer
- **Pre-conditions:** The actor is logged in. An item is currently on the shopping cart. The actor is viewing their shopping cart.
- **Description:** Allow the actor to delete items from his/her shopping cart.
- **Trigger:** The actor will click on the "Remove" button adjacent a certain item.
  The system responds by
  1. Deleting the item from the shopping cart for the particular actor.
- **Relevant Requirements:** None.
- **Post-Conditions:** Item is deleted from the shopping cart.

Alternate Courses of Action: None

Extensions: None

Exceptions: None

Concurrent Uses: None

Related Use Cases: HMTS_CU03_ViewCart

Decision Support:

Frequency: Medium – Function will be accessed 10 times per hour.

Criticality: Medium – Actors can still use system for its intended functionality without this function.

Risk: Medium – This function is explicit, but the mobile technique used to implement the system is new for the development team.

Constraints:

- **Usability**
  No previous training is needed.
- **Reliability**
  Mean time to failure – 1 failure for every 24 hours of operation.
- **Performance**

The procedure should take no more than 2 seconds.

- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system is to be done in Java, the Eclipse framework and Android framework.

Modification History

- **Owner:** Alex Marrero Pazos
- **Initiation Date:** 01/21/2016
- **Last Modified:** 03/23/2016

---

18. Fill Out Order Information for Customer

Use Case ID: HMTS_CU05_FillOutOrderInfo

Use Case Level: High-Level

Details:

- **Actor:** Customer
- **Pre-conditions:** The system is running. The actor has logged in the system, visualized menu, added dishes, sides, or drinks to a cart.
- **Description:** The system shall display a selection menu for the actor to choose a payment method. Different fields and buttons are displayed based on the actor's selection, requesting the information requiring for order placement.
- **Trigger:** The actor initiates an action by clicking on the "Check out" button.

  The system responds by

  1. The system displays a "Payment Methods" selection menu for the actor to choose a payment method, including the options "Pay by cash" and "Pay by debit/credit card".

2. If the actor selects “Pay by cash”, the system displays the fields “Shipping Address”, “Phone number (optional)”, “Delivery notes (optional)”, and the “Place Order” button.
3. When the actor clicks the “Place Order” button, the system attempts to place the order.
4. If the actor selects “Pay by debit/credit card”, the system displays the fields “Name on card”, “Card number”, “Expiration date”, “Security code”, “Billing address”, “Shipping address”, “Phone number (optional)”, “Deliver notes (optional)”, and the “Place Order” button.
5. When the actor clicks the “Place Order” button, the system attempts to verify the consistency of the addresses entered by the actor.

- **Relevant requirements:** None
- **Post-conditions:** The system enables the actor to chose a payment method, enter the required payment information, and request the order placement.

Alternative Courses of Action: None

Extensions: HMTS_CU07_PopAddressAlert

Exceptions:

- System failure before payment processing: order cancelled, information entered is lost.
- System failure after processing payment: order status is pending, payment confirmation information may be lost.
- Failure of User Interface before actor responds to system alert: order is stalled, times out after 5 minutes (HMTS_UM05_LoginTimeout).
- Failure of User Interface after order submission: order processed, confirmation may not be displayed, actor may be unable to further interact with system.

Concurrent Uses: None

Related Use Cases: HMTS_CU08_PlaceOrder, HMTS_CU06_VisualizePOC

Decision Support

Frequency: High – Performed for every time actors want to place an order. (300 times per hour)

Criticality: High – It is an important function for customers to place an order.

Risk: Low – We can store the data entered by the user into the system.

Constraints:

- **Usability**

  No previous training is needed.

- **Reliability**

  Mean Time to Failure – 1 failure for every 72 hours of operation is acceptable.

- **Performance**

  The payment information will be processed in 2 seconds.

- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system is to be done in Java, the Eclipse framework and Android framework.

Modification History

Owner: Raul Camarca

Initiation date: 01/21/2016

Date last modified: 03/10/2016

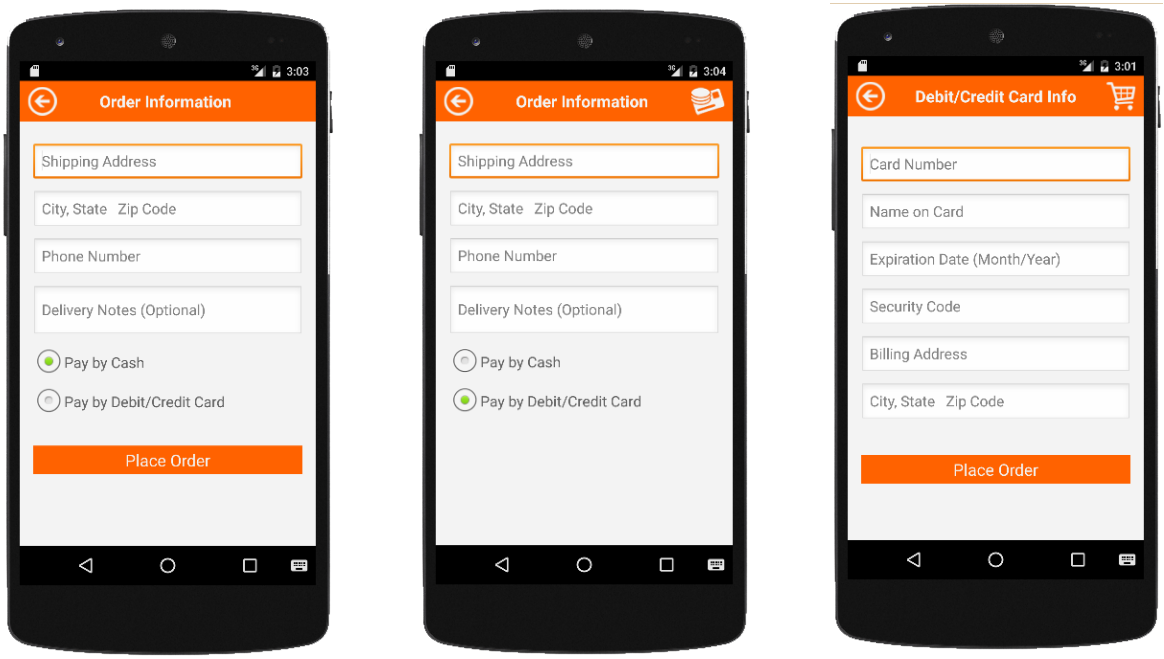


Figure 11.2.7. Fill out order info page for Customer.

19. Visualize POC (Purchase Order Confirmation) for Customer

Use Case ID: HMTS_CU06_Visualize POC

Use Case Level: High-Level

Details:

- **Actor:** Customer
- **Pre-conditions:** The actor has filled in order information and requested that the order be placed. The system has successfully processed the payment and placed the order.
- **Description:** The system visualizes a purchase order confirmation that the actor save or print, and allows the actor to access an order status section.
- **Trigger:** The actor clicks on "Place Order" and the system successfully completes the "place order" function.

  The system responds by

  1. The system will visualize the order number, date and time the order was placed, the dishes, sides, and drinks ordered, the chosen payment method (credit/debit card, cash), and the total payment charged (or to be charged, for cash).
  2. The system will provide buttons "Print" and "Visualize Order Status".

3. If the actor clicks on the “Print” button, a print dialogue will be visualized.
4. If the actor clicks on the “Visualize Order Status” button, the system will attempt to visualize the status for this order.

- **Relevant requirements:** None
- **Post-conditions:** The system displays a unique purchase order confirmation with relevant order details, the option to print the confirmation page, and the option to visualize the order status.

Alternative Courses of Action: None

Extensions: None

Exceptions: If the system experiences an electrical failure, the order will remain pending, although the system may not be able to display a confirmation to the actor and the actor may be unable to further interact with the system until the interface is working again.

Concurrent Uses: None

Related Use Cases: HMTS_CU05_FillOutOrderInfo, HMTS_CU08_PlaceOrder, HMTS_CU09_VisualizeStatus

---

Decision Support

Frequency: High – Performed for every time system successfully places an order (250 times per hour).

Criticality: Medium – The customer may want a receipt and enjoy the option to track his delivery.

Risk: Low – We will present to the customer information stored in the system and the ability to interact with it using two buttons.

Constraints:

- **Usability**

  No previous training is needed.
- **Reliability**

  Mean Time to Failure – 1 failure for every 24 hours of operation is acceptable.
- **Performance**

  The confirmation page will be created in 1 second. The print process will be triggered in 0.5 seconds. The visualize order status will be requested in 0.5 seconds.

- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system is to be done in Java, the Eclipse framework and Android framework.

Modification History

Owner: Raul Camarca

Initiation date: 01/21/2016

Date last modified: 04/08/2016

---

20. Pop Addresses Alert for Customer - Security

Use Case ID: HMTS_CU07_PopAddressAlert

Use Case Level: Functional Sub-use Case

Details:

- **Actor:** Customer
- **Pre-conditions:** The system is running. The actor has signed in the system, visualized menu, added dishes, sides, or drinks to a cart. The actor has requested to check out . The actor has selected "credit/debit card" as the form of payment and the system is displaying the required information fields that must be filled.
- **Description:** Actor fills out order information, enters a billing address different than the shipping address, and requests that the order be placed. If the system detects an inconsistency, it prompts the actor on whether to continue or go back and make changes.
- **Trigger:** The actor initiates an action by filling out order information and clicking on the "place order" button.

  The system responds by

  1. The system compares the two addresses.

2. If the system detects the two addresses are different, it displays the alert "Shipping and Billing addresses are different?" along with two buttons, "Place Order" and "Go Back".
3. If the actor clicks on the "Place Order" button, the system will attempt to place the order ignoring the inconsistency in the addresses.
4. If the actor clicks on the "Go Back" button, the system presents the order information filling page with the information the actor had previously entered.

- **Relevant requirements:** None
- **Post-conditions:** System will attempt to place an order given different shipping and billing addresses only if the actor confirms that this inconsistency is not the result of a human error.

Alternative Courses of Action: None.

Extensions: None.

Exceptions:

- System failure before payment processing: order cancelled, information entered is lost.
- System failure after processing payment: order status is pending, payment confirmation information may be lost.
- Failure of User Interface before actor responds to system alert: order is stalled, times out after 5 minutes (HMTS_UM05_LoginTimeout).
- Failure of User Interface after order submission: order processed, confirmation may not be displayed, actor may be unable to further interact with system.

Concurrent Uses: HMTS_CU05_FillOutOrderInfo

Related Use Cases: HMTS_UM05_LoginTimeout

Decision Support

Frequency: High – Performed for every time actors want to place an order. (300 times per hour)

Criticality: High – It's a important function for the system to verify order information.

Risk: Low – We can meet this requirement by implementing an address comparison algorithm.

Constraints:

- **Usability**

  No previous training is needed.

  Reliability

  Mean Time to Failure – 1 failure for every 24 hours of operation is acceptable.

- **Performance**

  The procedure of address comparison will be completed by the system in 1 second. The request to place order will be processed by the system in 2 seconds.

- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system is to be done in Java, the Eclipse framework and Android framework.

Modification History

Owner: Raul Camarca

Initiation date: 01/21/2016

Date last modified: 04/08/2016

---

21. Place Order for Customer

Use Case ID: HTMS_CU08_PlaceOrder

Use Case Level: High-Level

Details:

- **Actor:** Customer
- **Pre-conditions:** The actor has entered order information and requested that the order be placed.
- **Description:** The system shall place the order based on the selected payment method.
- **Trigger:** The actor initiates an action by clicking on the "Place Order" button.

The system responds by

1. The system processes the order based on the payment method specified by the actor.
2. If the system detects that the payment method is “Cash”, it will store the order as “pending”.
3. The system will visualize a purchase order confirmation.
4. If the system detects that the payment method is “Credit/Debit Card”, it will process the payment.
5. If the system processes the card successfully, it will visualize a purchase order confirmation.

- **Relevant requirements:** None
- **Post-conditions:** The system has a new order pending. The system will display relevant order information to the actor via the “visualize purchase order information” function.

Alternative Courses of Action: None

Extensions: None

Exceptions: If the system experiences an electrical failure before the payment is processed, the order will be cancelled automatically. If the system experiences an electrical failure after the payment is processed, the order will remain pending, although payment confirmation information may be lost.

Concurrent Uses: None

Related Use Cases: HMTS_CU05_FillOutOrderInfo, HMTS_CU06_VisualizePOC

Decision Support

Frequency: High – Performed for every time system successfully places an order (250 times per hour).

Criticality: High – the system must be able to process debit/credit card payments, or place the order if the actor chose to pay later with cash.

Risk: High – We must implement the credit/debit card payment processing with security measures including encryption.

Constraints:

- **Usability**

No previous training is needed.

Reliability

Mean Time to Failure – 1 failure for every 100 hours of operation is acceptable.

- **Performance**

  The debit/credit card order will be processed in 2 seconds. The order will be stored in the system in 1 second and the purchase order confirmation will be triggered in 1 second.

- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system is to be done in Java, the Eclipse framework and Android framework.

Modification History

Owner: Raul Camarca

Initiation date: 01/21/2016

Date last modified: 04/08/2016

---

22. Visualize Order Status for Customer

Use Case ID: HMTS_CU09_VisualizeStatus

Use Case Level: High-Level

Details:

- **Actor:** Customer
- **Pre-conditions:** The actor has successfully placed an order and has visualized the purchase order confirmation.
- **Description:** The system allows the actor to visualize the status of an order.
- **Trigger:** The actor initiates an action by clicking on the "Visualize Order Status" button.

The system responds by

1. The system visualizes the order number and the status (“Pending”, “In Transit”, “Delayed”, “Delivered”, “Canceled”, “Refunded”).
2. The system includes a “Refresh” button.
3. If the actor clicks on the “Refresh” button, the system will visualize the most up-to-date status of the order.

- **Relevant requirements:** None
- **Post-conditions:** The system shows the status of a given order.

Alternative Courses of Action: None.

Extensions: None

Exceptions: If the system experiences a failure after the actor clicks on “Visualize Order Status”, it will be unable to display order status information. If the system experiences a failure after the actor clicks on “Refresh”, it will be unable to display up-to-date order status information.

Concurrent Uses: None

Related Use Cases: HMTS_CU06_VisualizePOC

Decision Support

Frequency: Low – Performed for every time actors want to track the status of their order. (30 times per hour)

Criticality: Low – it is of some usefulness to the customer to know the order status.

Risk: Low – We can implement this functionality by retrieving data stored in the system and visualizing it for the customer.

Constraints:

- **Usability**

  No previous training is needed.
- **Reliability**

  Mean Time to Failure – 1 failure for every 12 hours of operation is acceptable.
- **Performance**

  The status will be displayed in 1 second and refreshed in 1 second.
- **Supportability**

The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system is to be done in Java, the Eclipse framework and Android framework.

Modification History

Owner: Raul Camarca

Initiation date: 01/21/2016

Date last modified: 04/08/2016

---

23. Visualize Orders for Delivery man

Use Case ID: HMTS_DE01_VisualizeOrders

Use Case Level: High-Level

Details:

- **Actor:** Delivery man
- **Pre-conditions:** The system is running. The actor has signed in the system.
- **Description:** The system shall visualize an order list containing the information about the orders he is responsible. For each order, it contains the customer name, order time, delivery address and total price.
- **Trigger:** The actor initiates an action by clicking "View my orders" item on the navigation bar.

  The system responds by

  1. The system visualizes all the orders' information for the current actor. If there is not any order, it will only show the titles of each column in the list.
- **Relevant requirements:** None
- **Post-conditions:** The actors can see all the orders they have now and perform some operations on the orders list, for example, view details of an order.

Alternative Courses of Action: None

Extensions: HMTS_DE02_VisualizeDetailedOrder

Exceptions: None

Concurrent Uses: None

Related Use Cases: HMTS_DE02_VisualizeDetailedOrder

Decision Support

Frequency: Medium – Performed for every time a delivery man want to view his orders. (30 times per day)

Criticality: Medium – It's a useful function for a delivery man to manage his orders.

Risk: Medium – This function is explicit, but the mobile technique used to implement the system is new for the development team.

Constraints:

- **Usability**

  No previous training is needed.

- **Reliability**

  Mean Time to Failure – 1 failure for every 24 hours of operation is acceptable.

- **Performance**

  The orders information will be shown in 2 seconds.

- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system is to be done in Java, the Eclipse framework and Android framework.

Modification History

Owner: Weitong Liu

Initiation date: 01/24/2016

Date last modified: 04/08/2016

24. Visualize Detailed Order for Delivery man

Use Case ID: HMTS_DE02_VisualizeDetailedOrder

Use Case Level: High-Level

Details:

- **Actor:** Delivery man
- **Pre-conditions:** The system is running. The actor has signed in the system and entered the "Visualize Orders" page.
- **Description:** The system allow Delivery man to view detailed order information. For an order, it contains the customer name, order time, delivery address, total price, items in the order, and customer's message.
- **Trigger:** The actor initiates an action by clicking "View order information" item on the "Visualize Orders" page.

  The system responds by

  1. The system visualizes the corresponding detailed order information for the current actor.
- **Relevant requirements:** None
- **Post-conditions:** The actor can see all the detailed information and perform some operations on the order, for example, process the order.

Alternative Courses of Action: None

Extensions: HMTS_DE03_ProcessOrder

Exceptions: None

Concurrent Uses: None

Related Use Cases: HMTS_DE01_VisualizeOrders, HMTS_DE03_ProcessOrder

---

Decision Support

Frequency: Medium – Performed for every time a delivery man want to view detailed order information. (30 times per day)

Criticality: Medium – It's a useful function for a delivery man to manage his orders.

Risk: Medium – This function is explicit, but the mobile technique used to implement the system is new for the development team.

Constraints:

- **Usability**

  No previous training is needed.

- **Reliability**

  Mean Time to Failure – 1 failure for every 24 hours of operation is acceptable.

- **Performance**

  The detailed order information will be shown in 2 seconds.

- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.

- **Implementation**

  The implementation of the system is to be done in Java, the Eclipse framework and Android framework.

Modification History

Owner: Weitong Liu

Initiation date: 01/24/2016

Date last modified: 04/08/2016

---

25. Process Order for Delivery man

Use Case ID: HMTS_DE03_ProcessOrder

Use Case Level: High-Level

Details:

- **Actor:** Delivery man
- **Pre-conditions:** The system is running. The actor has signed in the system and entered the “View order information” page.
- **Description:** The system allow Delivery man to update the delivery status of the orders. The delivery status of an order may be “delivered”, “undelivered (not at home)” or “undelivered (rejected)”.

- **Trigger:** The actor initiates an action by selecting an item in the dropdown list of order status, and clicking “confirm” button on the “View order information” page.

  The system responds by

  1. The system updates selected order status and shows the result (“update succeeded” or “update failed”) for the current actor.
- **Relevant requirements:** None
- **Post-conditions:** Actors can review this order, or return to the order list to continue with another order.

Alternative Courses of Action: None

Extensions: None

Exceptions: None

Concurrent Uses: None

Related Use Cases: HMTS_DE02_VisualizeDetailedOrder

Decision Support

Frequency: Medium – Performed for every time a delivery man want to update the delivery status of an order. (20 times per day)

Criticality: Medium – It’s a useful function for a delivery man to manage his orders.

Risk: Medium – This function is explicit, but the mobile technique used to implement the system is new for the development team.

Constraints:

- **Usability**

  No previous training is needed.
- **Reliability**

  Mean Time to Failure – 1 failure for every 24 hours of operation is acceptable.
- **Performance**

  The operation will be executed in 2 seconds.
- **Supportability**

  The application will rely on the Java platform (version 7 or later) so it can be ported to any environment where Java can be installed.
- **Implementation**

The implementation of the system is to be done in Java, the Eclipse framework and Android framework.

Modification History

Owner: Weitong Liu

Initiation date: 01/24/2016

Date last modified: 04/08/2016

11.3. Appendix C – User Interface Designs

In this section we show all the user interfaces designs for the use cases that are going to be implemented for the mobile application as well as for the web application.

11.3.1. Mobile App

In this section we show all the user interfaces for mobile applications.

1. Registration Page

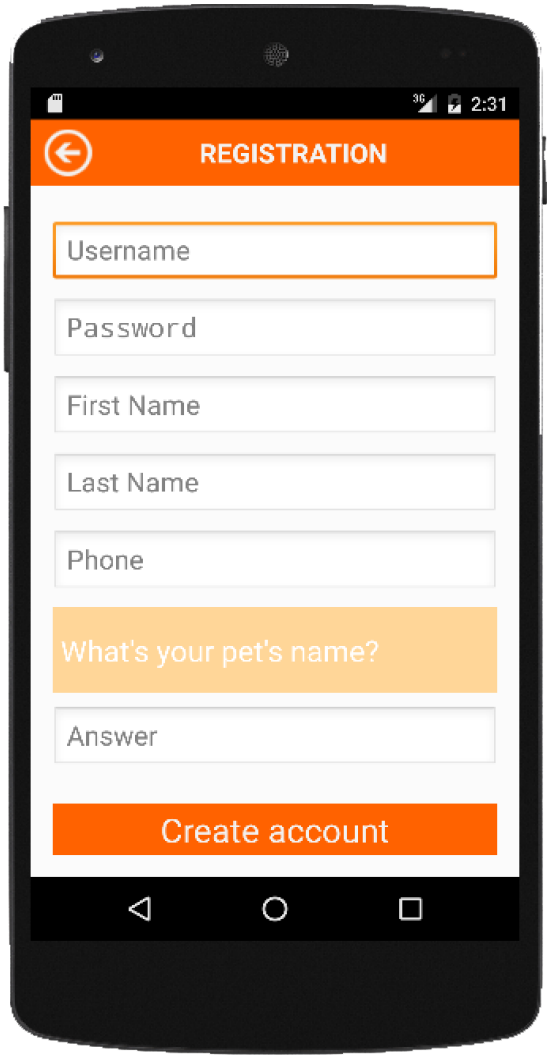


Figure 11.3.1. Mobile Registration page for customers.

2. Login Page

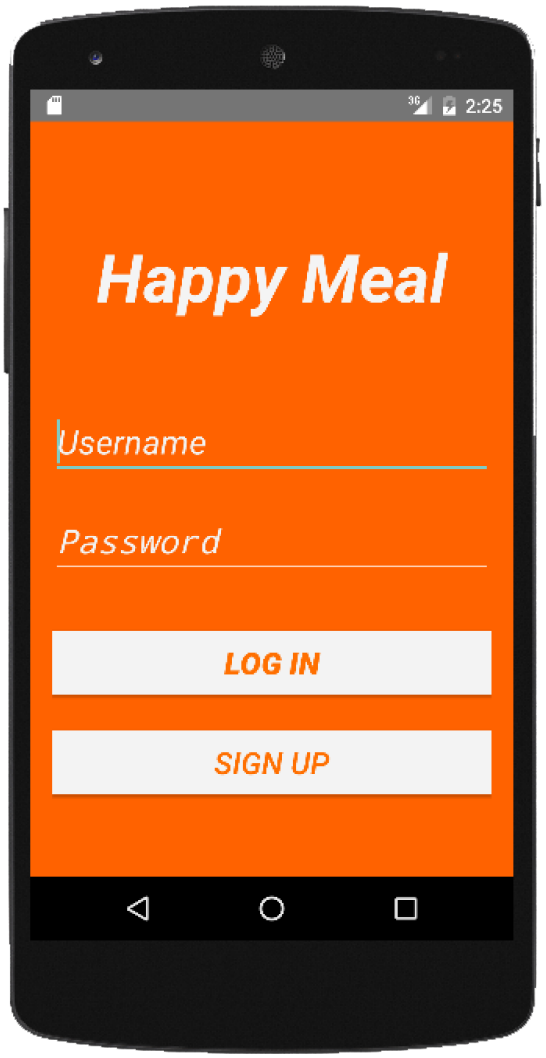


Figure 11.3.2. Mobile Login page for customers.

3. Visualize Menu Page

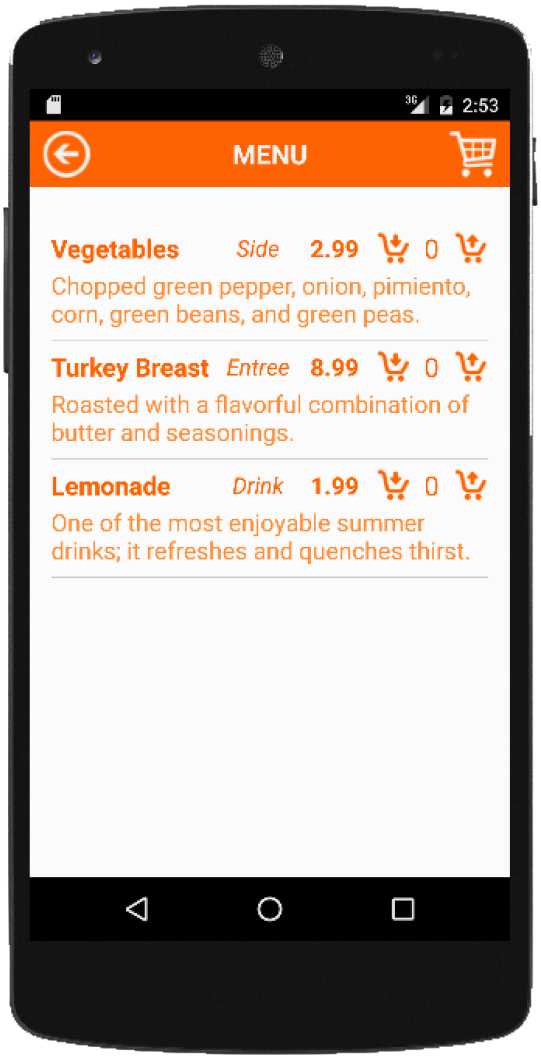


Figure 11.3.3. Mobile Visualize menu page for customers.

4. Visualize Cart Page

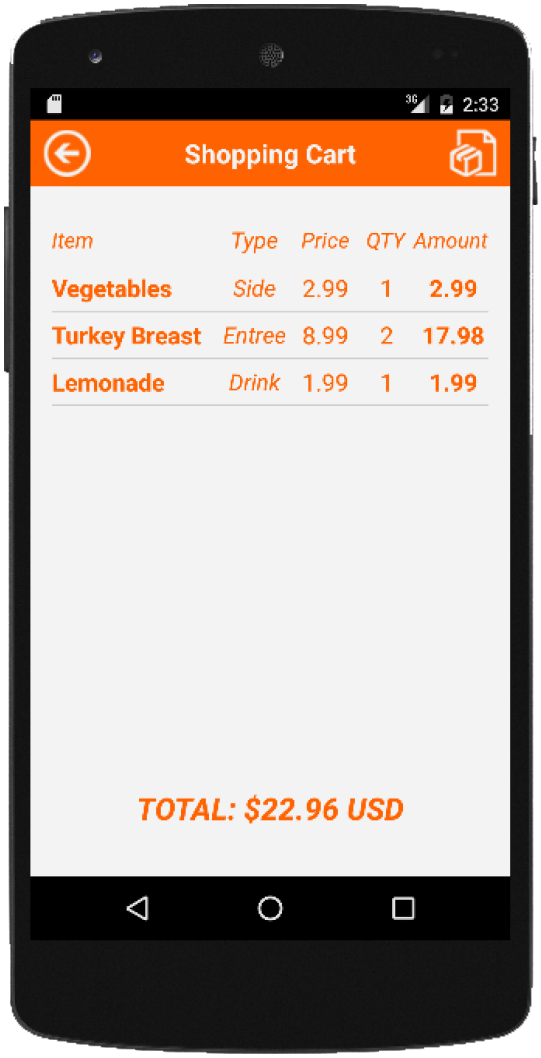


Figure 11.3.4. Mobile Visualize shopping cart page for customers.

5. Fill Order Info Page

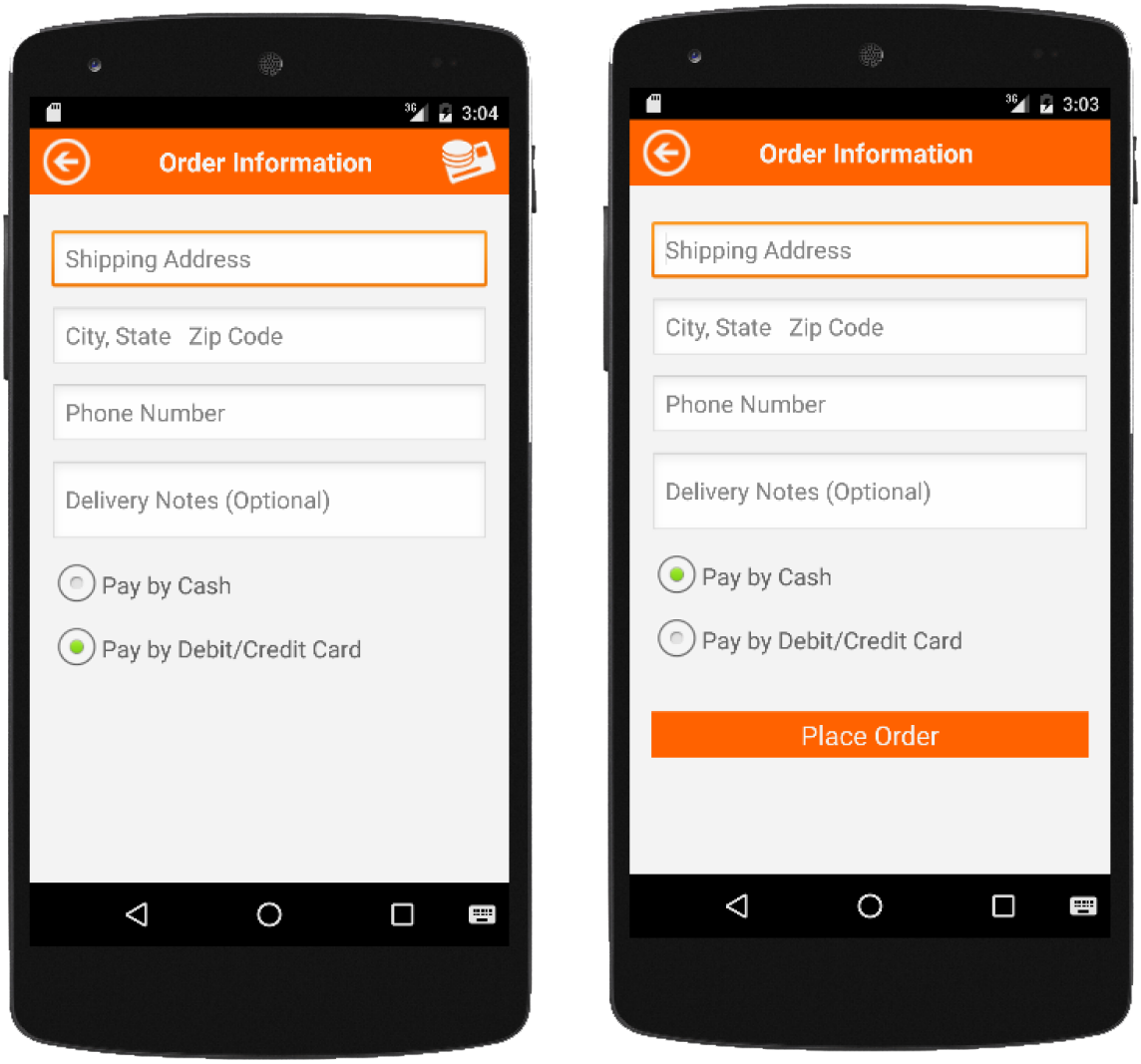


Figure 11.3.5. Mobile Fill out order (payment with cash) page for customers.

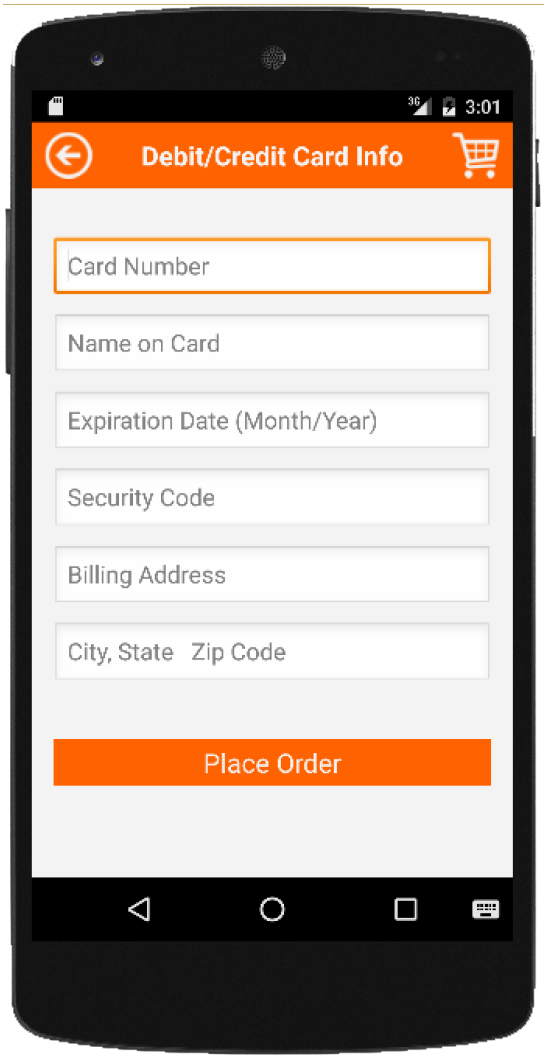


Figure 11.3.6. Mobile Fill out order (payment with credit/debit card) page for customers.

11.3.2. Web App

In this section we show all the user interfaces for web applications.

1. Registration Page

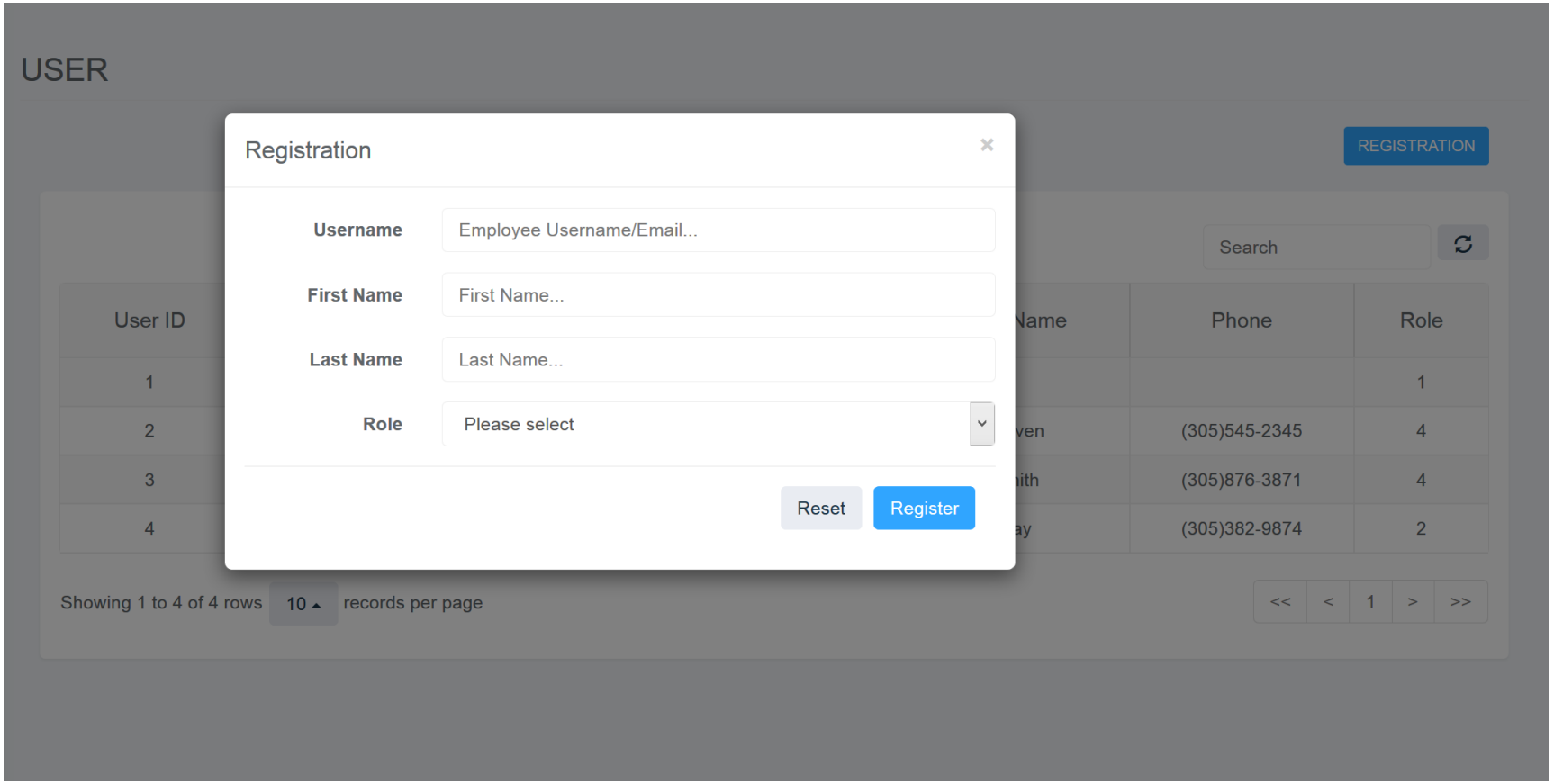


Figure 11.3.7. Web Registration page for manager.

2. Login Page

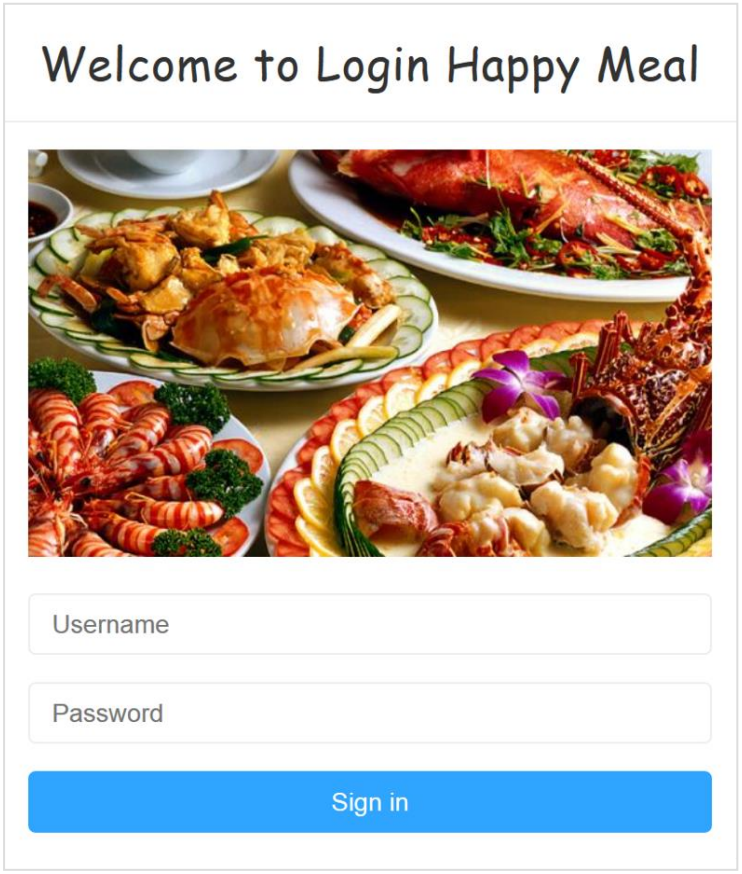


Figure 11.3.8. Web Login page for manager.

3. Home Page

➢ For Manager

Figure 11.3.9. Web Home page for manager.

➢ Staff

Figure 11.3.10. Web Home page for staff.

4. Visualize Products Page

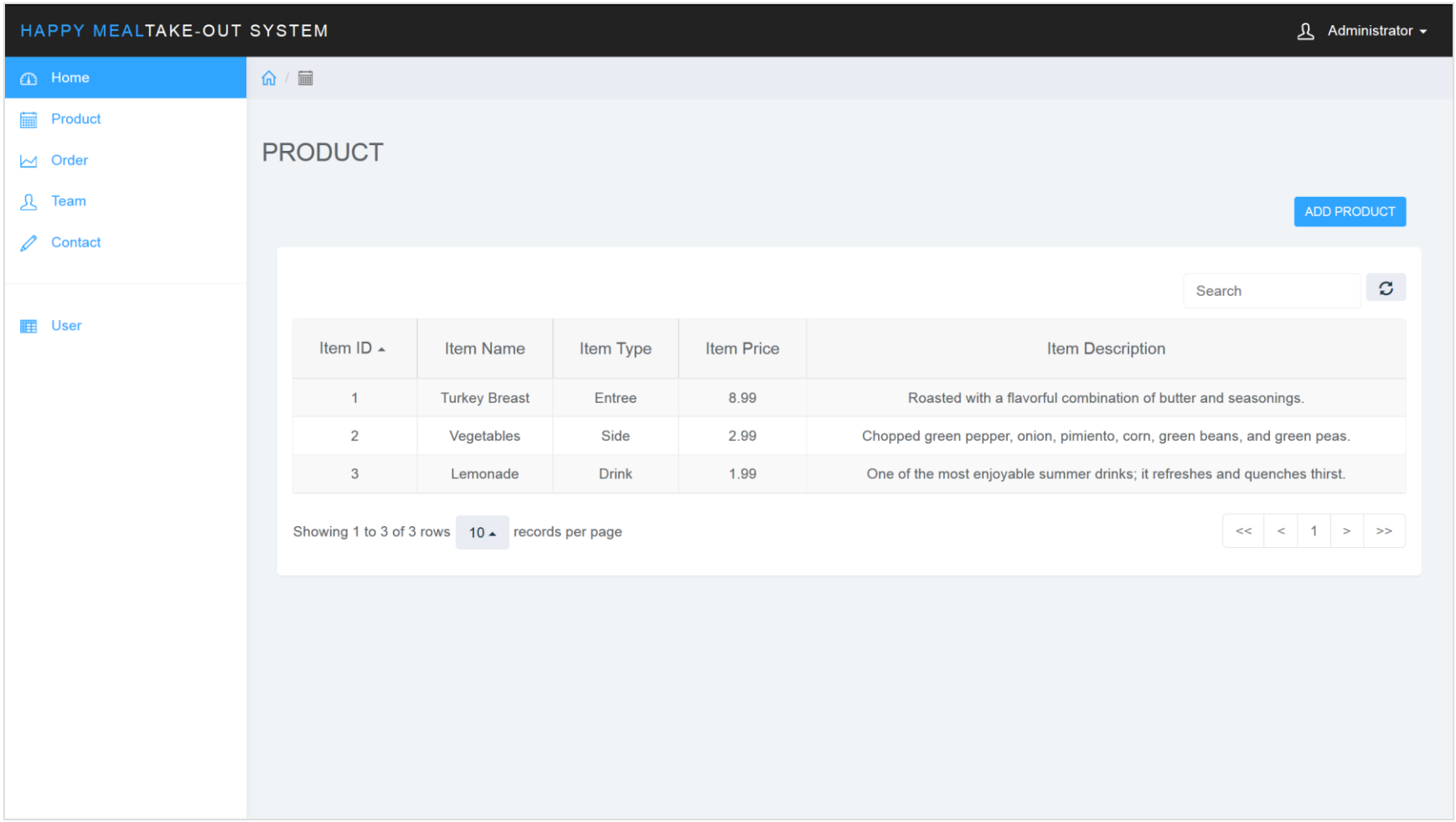


Figure 11.3.11. Web Visualize products page for staff.

5. Add Product Page

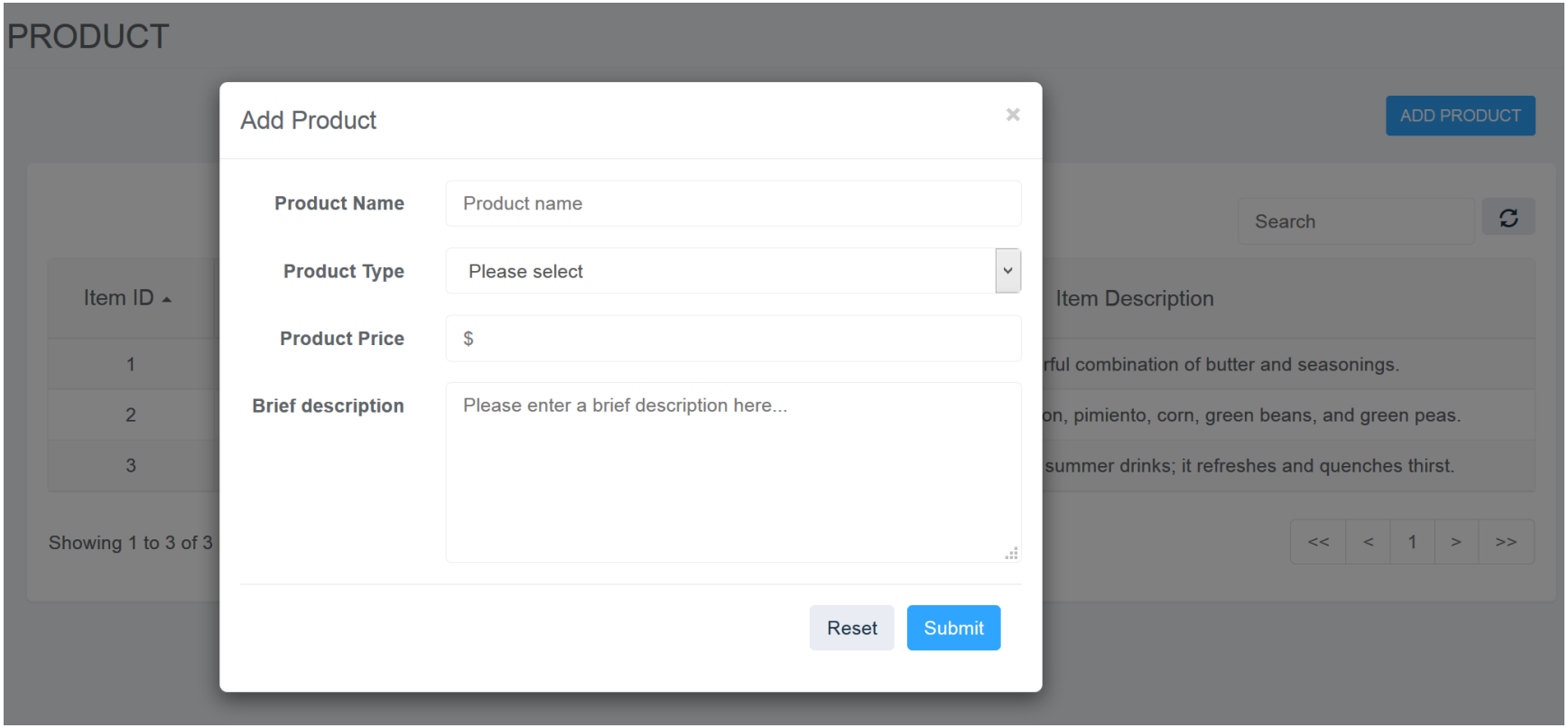


Figure 11.3.12. Web Add product page for staff.

11.4. Appendix D – Detail Class Diagrams

In this section we provide detail class diagrams showing attributes and methods for each class.

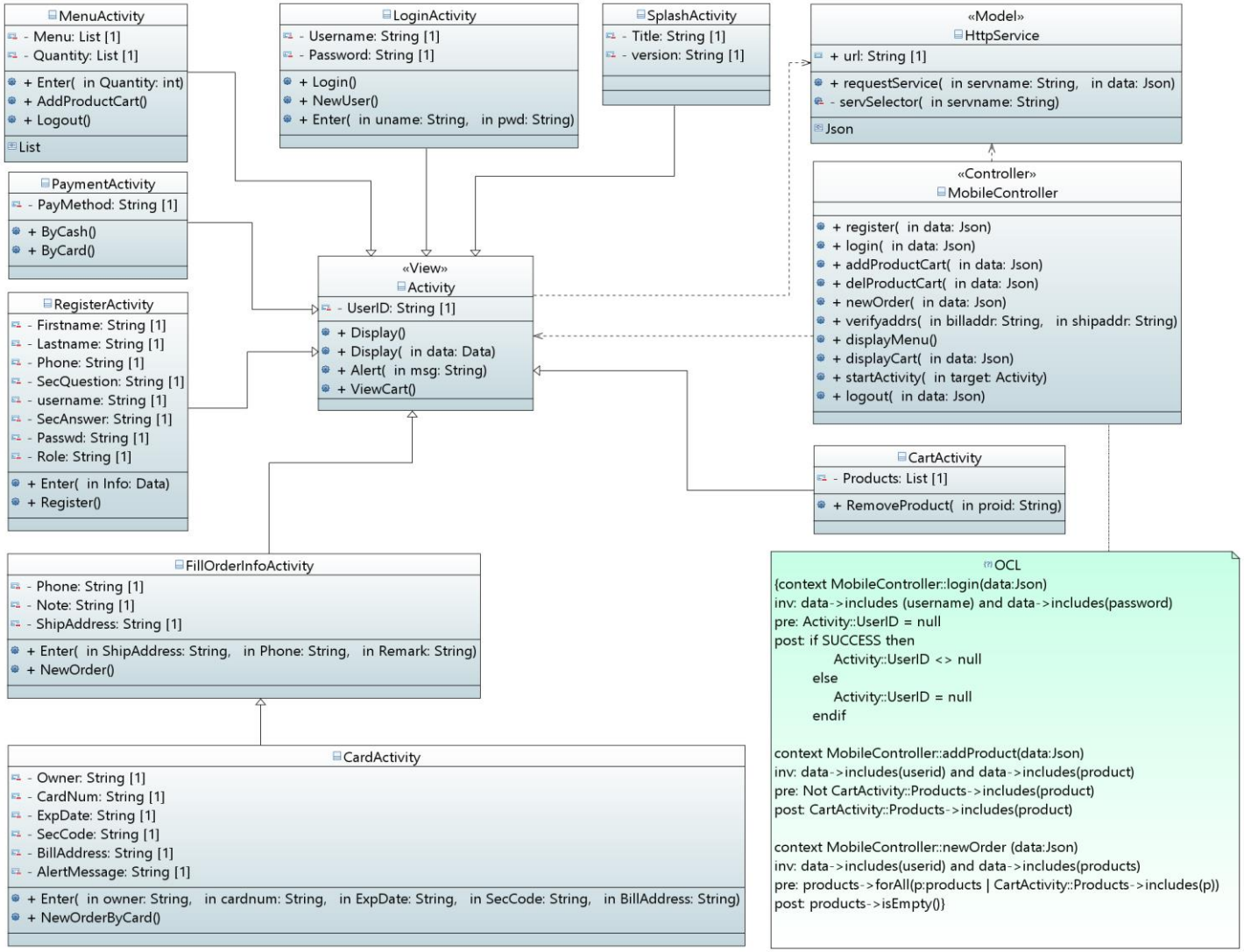


Figure 11.4.1. Detailed class diagram for Native Mobile Application subsystem.

In Figure 11.4.1, we describe the detailed classes with attributes, methods and OCL for mobile client subsystem and relationships among the classes.

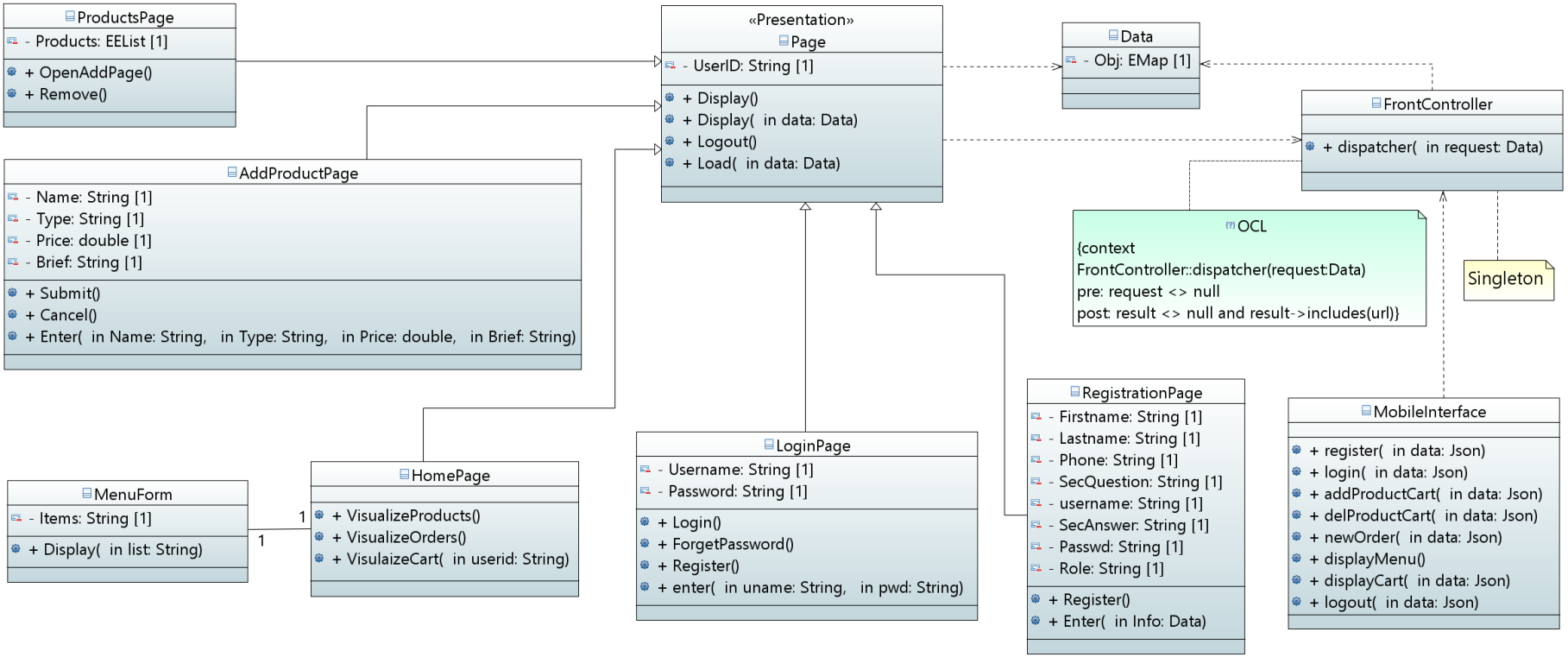


Figure 11.4.2. Detailed class diagram for Presentation subsystem.

In Figure 11.4.2, we describe the detailed classes with attributes, methods and OCL for presentation subsystem and relationships among the classes.

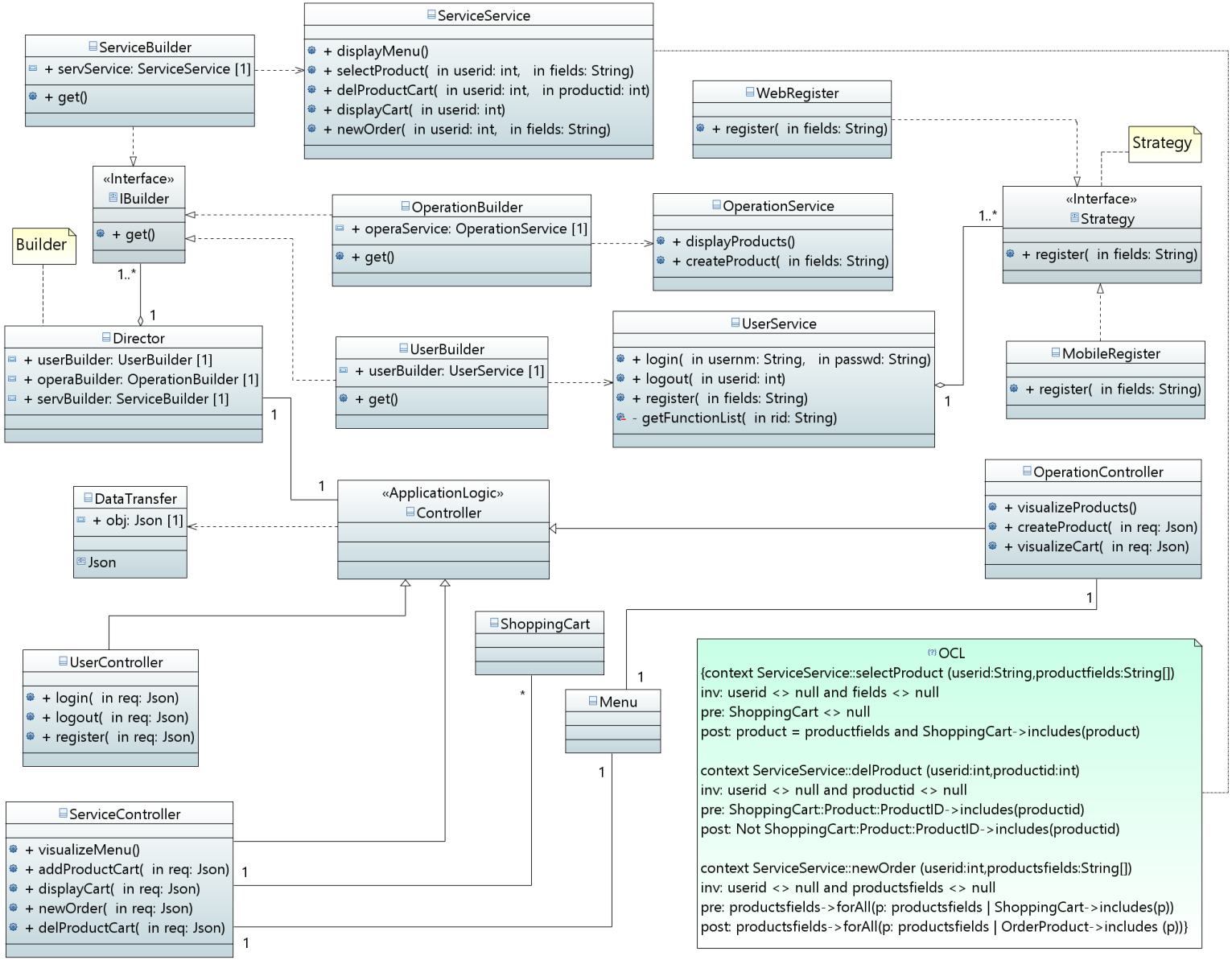


Figure 11.4.3. Detailed class diagram for ApplicationLogic subsystem.

In Figure 11.4.3, we describe the detailed classes with attributes, methods and OCL for application logic subsystem and relationships among the classes.

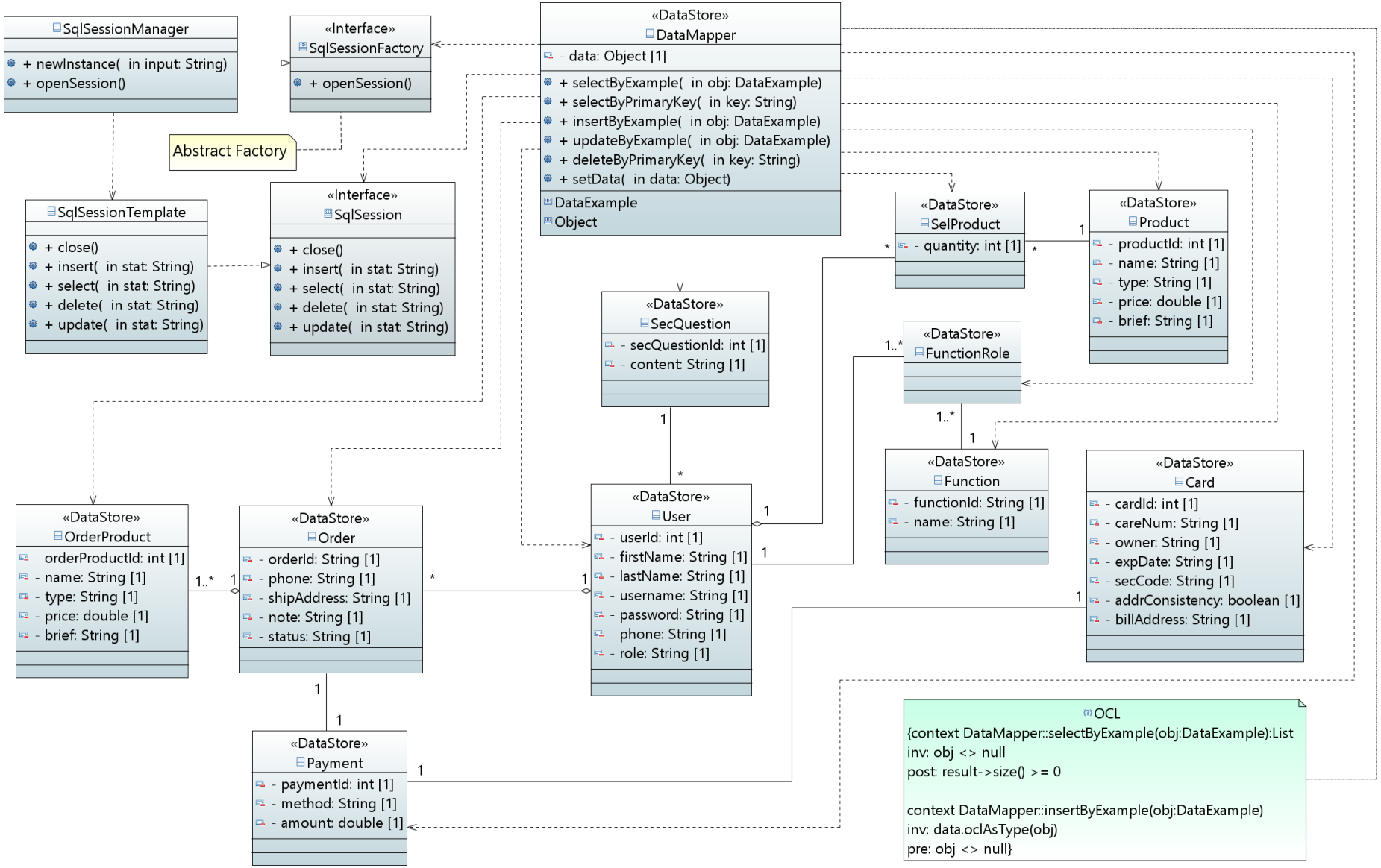


Figure 11.4.4. Detailed class diagram for DataStore subsystem.

In Figure 11.4.4, we describe the detailed classes with attributes, methods and OCL for data store subsystem and relationships among the classes.

11.5. Appendix E – Class Interfaces (code) for Subsystem(s)

In this section we present the class interface code for the main control objects in each subsystem. All the class interfaces generated by Javadoc for both web and mobile ends are available at the following websites:

- For mobile: *http://users.cs.fiu.edu/~wwang035/hmts_doc/mobile*
- For web: *http://users.cs.fiu.edu/~wwang035/hmts_doc/web*

11.6. Appendix F – Documented Code for Test Driver

In this section, we talk about the documented code for test drivers for three types of testing, unit testing, subsystem testing and system testing.

11.6.1. Unit Test Codes

- Test Suite

```
package unit;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

import unit.testcase.service.OperationServiceTest;
import unit.testcase.service.ServiceServiceTest;
import unit.testcase.service.UserServiceTest;


/**
 * The Class TestSuite.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@RunWith(Suite.class)
@SuiteClasses({OperationServiceTest.class
        , UserServiceTest.class
        , ServiceServiceTest.class})

public class TestSuite  {}
```

- UserServiceTest

```
package unit.testcase.service;

import static org.junit.Assert.*;

import java.util.ArrayList;
import java.util.List;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
```

```
import org.mockito.Mockito;
import org.mockito.MockitoAnnotations;
import org.mockito.Spy;
import org.mockito.runners.MockitoJUnitRunner;

import edu.fiu.hmts.dao.hmts_repos.FunctionRoleMapper;
import edu.fiu.hmts.dao.hmts_repos.UserMapper;
import edu.fiu.hmts.domain.hmts_repos.FunctionRoleExample;
import edu.fiu.hmts.domain.hmts_repos.FunctionRoleKey;
import edu.fiu.hmts.domain.hmts_repos.User;
import edu.fiu.hmts.domain.hmts_repos.UserExample;
import edu.fiu.hmts.service.impl.UserService;
import edu.fiu.hmts.util.registerstrategy.MobileRegister;
import edu.fiu.hmts.util.registerstrategy.WebRegister;


/**
 * The Class UserServiceTest.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@RunWith(MockitoJUnitRunner.class)
public class UserServiceTest {

	/** The user service. */
	@Spy
	@InjectMocks
	UserService userService;

	/** The user mapper. */
	@Mock
	UserMapper userMapper;

	/** The function role mapper. */
	@Mock
	FunctionRoleMapper functionRoleMapper;

	/** The web register. */
	@Mock
	WebRegister webRegister;

	/** The mobile register. */
	@Mock
	MobileRegister mobileRegister;

	/** The mob user s. */
	User userNU, webUserS, mobUserS;
```

```
/** The user list f. */
List<User> webUserList, mobUserList, userListF;

/** The function role keys2. */
List<FunctionRoleKey> functionRoleKeys, functionRoleKeys2;

/**
 * Sets the up.
 *
 * @throws Exception
 *             the exception
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Before
public void setUp() throws Exception {
        MockitoAnnotations.initMocks(this);

        userNU = new User();
        userNU.setUserId(-1L);

        webUserS = new User();
        webUserS.setUsername("peter@gmail.com");
        webUserS.setPassword("123456");
        webUserS.setFirstName("Peter");
        webUserS.setLastName("John");
        webUserS.setPhone("(305)837-4827");
        webUserS.setRole("2");

        mobUserS = new User();
        mobUserS.setUsername("david@yahoo.com");
        mobUserS.setPassword("123456");
        mobUserS.setFirstName("David");
        mobUserS.setLastName("Chris");
        mobUserS.setPhone("(305)233-9483");
        mobUserS.setSecQuestionId(1);
        mobUserS.setSecAnswer("Paul");
        mobUserS.setRole("4");

        userListF = new ArrayList<>();
        mobUserList = new ArrayList<>();
        webUserList = new ArrayList<>();
        webUserList.add(webUserS);
        mobUserList.add(mobUserS);

        functionRoleKeys = new ArrayList<>();
```

```
            FunctionRoleKey functionRoleKey1 = new FunctionRoleKey();
            functionRoleKey1.setFunctionId(1);
            functionRoleKeys.add(functionRoleKey1);
            FunctionRoleKey functionRoleKey2 = new FunctionRoleKey();
            functionRoleKey2.setFunctionId(2);
            functionRoleKeys.add(functionRoleKey2);

            functionRoleKeys2 = new ArrayList<>();
        }

        /**
         * Test web register success.
         *
         * @author  Wenbo Wang
         * @version 1.0, April 2016
         */
        @Test
        public final void testWebRegisterSuccess() {
            Mockito.when(webRegister.register(Mockito.any(String.class),
Mockito.any(String.class)
                    , Mockito.any(String.class), Mockito.any(String.class),
Mockito.any(String.class)
                    , Mockito.any(String.class), Mockito.any(int.class),
Mockito.any(String.class))).thenReturn(webUserS);
            User user = userService.register("peter@gmail.com", "123456", "Peter"
                    , "John", "(305)837-4827", "2", -1, "", 0);
            assertEquals("peter@gmail.com", user.getUsername());
            assertEquals("123456", user.getPassword());
            assertEquals("Peter", user.getFirstName());
            assertEquals("John", user.getLastName());
            assertEquals("(305)837-4827", user.getPhone());
            assertEquals("2", user.getRole());
        }

        /**
         * Test web register no username.
         *
         * @author  Wenbo Wang
         * @version 1.0, April 2016
         */
        @Test
        public final void testWebRegisterNoUsername() {
            Mockito.when(webRegister.register(Mockito.any(String.class),
Mockito.any(String.class)
                    , Mockito.any(String.class), Mockito.any(String.class),
Mockito.any(String.class)
                    , Mockito.any(String.class), Mockito.any(int.class),
Mockito.any(String.class))).thenReturn(userNU);
```

```
		User user = userService.register("", "123456", "Peter"
				, "John", "(305)837-4827", "2", -1, "", 0);
		assertEquals(null, user.getUsername());
		assertEquals(null, user.getPassword());
		assertEquals(null, user.getFirstName());
		assertEquals(null, user.getLastName());
		assertEquals(null, user.getPhone());
		assertEquals(null, user.getRole());
	}

	/**
	 * Test mobile register success.
	 *
	 * @author  Wenbo Wang
	 * @version 1.0, April 2016
	 */
	@Test
	public final void testMobileRegisterSuccess() {
		Mockito.when(mobileRegister.register(Mockito.any(String.class),
Mockito.any(String.class)
				, Mockito.any(String.class), Mockito.any(String.class),
Mockito.any(String.class)
				, Mockito.any(String.class), Mockito.any(int.class),
Mockito.any(String.class))).thenReturn(mobUserS);
		User user = userService.register("david@yahoo.com", "123456", "David",
"Chris", "(305)233-9483"
				, "4", 1, "Paul", 1);
		assertEquals("david@yahoo.com", user.getUsername());
		assertEquals("123456", user.getPassword());
		assertEquals("David", user.getFirstName());
		assertEquals("Chris", user.getLastName());
		assertEquals("(305)233-9483", user.getPhone());
		assertEquals("4", user.getRole());
		assertEquals(1, user.getSecQuestionId(), 1);
		assertEquals("Paul", user.getSecAnswer());
	}

	/**
	 * Test mobile register no username.
	 *
	 * @author  Wenbo Wang
	 * @version 1.0, April 2016
	 */
	@Test
	public final void testMobileRegisterNoUsername() {
		Mockito.when(mobileRegister.register(Mockito.any(String.class),
Mockito.any(String.class)
```

```
                , Mockito.any(String.class), Mockito.any(String.class),
Mockito.any(String.class)
                , Mockito.any(String.class), Mockito.any(int.class),
Mockito.any(String.class))).thenReturn(userNU);
        User user = userService.register("", "123456", "David", "Chris", "(305)233-9483"
                , "4", 1, "Paul", 1);
        assertEquals(null, user.getUsername());
        assertEquals(null, user.getPassword());
        assertEquals(null, user.getFirstName());
        assertEquals(null, user.getLastName());
        assertEquals(null, user.getPhone());
        assertEquals(null, user.getRole());
    }

    /**
     * Test register invalid type.
     *
     * @author  Wenbo Wang
     * @version 1.0, April 2016
     */
    @Test
    public final void testRegisterInvalidType() {
        Mockito.when(mobileRegister.register(Mockito.any(String.class),
Mockito.any(String.class)
                , Mockito.any(String.class), Mockito.any(String.class),
Mockito.any(String.class)
                , Mockito.any(String.class), Mockito.any(int.class),
Mockito.any(String.class))).thenReturn(userNU);
        User user = userService.register("", "123456", "David", "Chris", "(305)233-9483"
                , "4", 1, "Paul", -1);
        assertEquals(null, user.getUsername());
        assertEquals(null, user.getPassword());
        assertEquals(null, user.getFirstName());
        assertEquals(null, user.getLastName());
        assertEquals(null, user.getPhone());
        assertEquals(null, user.getRole());
    }

    /**
     * Test login web success.
     *
     * @author  Wenbo Wang
     * @version 1.0, April 2016
     */
    @Test
    public final void testLoginWebSuccess() {
        try {
```

```
        Mockito.when(userMapper.selectByExample(Mockito.any())).thenReturn(webUserList);
                Mockito.doReturn("1,2,").when(userService).rolesControl(Mockito.any());
                User user = userService.login("peter@gmail.com", "123456", 0);
                assertEquals("peter@gmail.com", user.getUsername());
                assertEquals("123456", user.getPassword());
                assertEquals("2", user.getRole());
                assertEquals("1,2,", user.getTag());
            }catch(Exception e){
                e.printStackTrace();
            }
        }

        /**
         * Test login web failed.
         *
         * @author  Wenbo Wang
         * @version 1.0, April 2016
         */
        @Test
        public final void testLoginWebFailed() {

        Mockito.when(userMapper.selectByExample(Mockito.any())).thenReturn(userListF);
            User user = userService.login("peter@gmail.com", "111111", 0);
            assertEquals(-1L, user.getUserId(), 1);
            assertEquals(null, user.getUsername());
            assertEquals(null, user.getPassword());
            assertEquals(null, user.getRole());
        }

        /**
         * Test login mob success.
         *
         * @author  Wenbo Wang
         * @version 1.0, April 2016
         */
        @Test
        public final void testLoginMobSuccess() {

        Mockito.when(userMapper.selectByExample(Mockito.any(UserExample.class))).thenRet
urn(mobUserList);
            Mockito.doReturn("").when(userService).rolesControl(Mockito.any());
            User user = userService.login("david@yahoo.com", "123456", 1);
            assertEquals("david@yahoo.com", user.getUsername());
            assertEquals("123456", user.getPassword());
            assertEquals("4", user.getRole());
        }
```

```
	/**
	 * Test login mob failed.
	 *
	 * @author  Wenbo Wang
	 * @version 1.0, April 2016
	 */
	@Test
	public final void testLoginMobFailed() {

	Mockito.when(userMapper.selectByExample(Mockito.any(UserExample.class))).thenRet
urn(userListF);
		User user = userService.login("david@yahoo.com", "111111", 1);
		assertEquals(-1L, user.getUserId(), 1);
		assertEquals(null, user.getUsername());
		assertEquals(null, user.getPassword());
		assertEquals(null, user.getRole());
	}

	/**
	 * Test login invalid type.
	 *
	 * @author  Wenbo Wang
	 * @version 1.0, April 2016
	 */
	@Test
	public final void testLoginInvalidType() {

	Mockito.when(userMapper.selectByExample(Mockito.any())).thenReturn(userListF);
		User user = userService.login("peter@gmail.com", "111111", -1);
		assertEquals(-1L, user.getUserId(), 1);
		assertEquals(null, user.getUsername());
		assertEquals(null, user.getPassword());
		assertEquals(null, user.getRole());
	}

	/**
	 * Test logout.
	 *
	 * @author  Wenbo Wang
	 * @version 1.0, April 2016
	 */
	@Test
	public final void testLogout() {
		boolean res = userService.logout(1);
		assertEquals(true, res);
	}

	/**
```

```
     * Test roles control.
     *
     * @author  Wenbo Wang
     * @version 1.0, April 2016
     */
    @Test
    public final void testRolesControl() {

    Mockito.when(functionRoleMapper.selectByExample(Mockito.any(FunctionRoleExampl
e.class))).thenReturn(functionRoleKeys);
            String res = userService.rolesControl(webUserS);
            assertEquals("1,2,", res);
    }

    /**
     * Test display users success.
     *
     * @author  Wenbo Wang
     * @version 1.0, April 2016
     */
    @Test
    public final void testDisplayUsersSuccess() {

    Mockito.when(userMapper.selectByExample(Mockito.any())).thenReturn(webUserList);
            List<User> list = userService.displayUsers();
            assertEquals(1, list.size());
            assertEquals("peter@gmail.com", list.get(0).getUsername());
            assertEquals("Peter", list.get(0).getFirstName());
            assertEquals("John", list.get(0).getLastName());
            assertEquals("(305)837-4827", list.get(0).getPhone());
            assertEquals("2", list.get(0).getRole());
    }

    /**
     * Test display users no user.
     *
     * @author  Wenbo Wang
     * @version 1.0, April 2016
     */
    @Test
    public final void testDisplayUsersNoUser() {

    Mockito.when(userMapper.selectByExample(Mockito.any())).thenReturn(userListF);
            List<User> list = userService.displayUsers();
            assertEquals(0, list.size());
    }
}
```

- OperationServiceTest

```
package unit.testcase.service;

import static org.junit.Assert.*;

import java.util.ArrayList;
import java.util.List;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.MockitoAnnotations;
import org.mockito.runners.MockitoJUnitRunner;

import edu.fiu.hmts.dao.hmts_repos.OrderMapper;
import edu.fiu.hmts.dao.hmts_repos.ProductMapper;
import edu.fiu.hmts.domain.hmts_repos.Order;
import edu.fiu.hmts.domain.hmts_repos.Product;
import edu.fiu.hmts.domain.hmts_repos.ProductExample;
import edu.fiu.hmts.service.impl.OperationService;

/**
 * The Class OperationServiceTest.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@RunWith(MockitoJUnitRunner.class)
public class OperationServiceTest {

	/** The opera service. */
	@InjectMocks
	OperationService operaService;

	/** The product mapper. */
	@Mock
	ProductMapper productMapper;

	/** The order mapper. */
	@Mock
	OrderMapper orderMapper;

	/** The pro list display. */
	List<Product> proListDisplay = new ArrayList<>();
```

```
/** The order list. */
List<Order> orderList = new ArrayList<>();

/** The product. */
Product product = new Product();

/**
 * Sets the up.
 *
 * @throws Exception
 *             the exception
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Before
public void setUp() throws Exception {
        MockitoAnnotations.initMocks(this);

        Product product1 = new Product();
        product1.setName("Turkey breast");
        product1.setType("Dish");
        product1.setPrice(8.99);
        proListDisplay.add(product1);

        Product product2 = new Product();
        product2.setName("Vegetables");
        product2.setType("Side");
        product2.setPrice(2.99);
        proListDisplay.add(product2);

        Product product3 = new Product();
        product3.setName("Lemonade");
        product3.setType("Drink");
        product3.setPrice(1.99);
        proListDisplay.add(product3);

        product.setName("Pizza");
        product.setType("Side");
        product.setPrice(10.99);
        product.setBrief("Pork, Onions and Cheese.");

        Order order = new Order();
        order.setPhone("(305)474-2378");
        order.setShipAddress("123 NW 14th St, Miami FL 33100");
        order.setStatus("processing");
        order.setOrderId(5L);
```

```
            order.setUserId(3L);
            orderList.add(order);
        }

        /**
         * Test display products.
         *
         * @author  Wenbo Wang
         * @version 1.0, April 2016
         */
        @Test
        public void testDisplayProducts() {

        Mockito.when(productMapper.selectByExample(Mockito.any(ProductExample.class))).t
henReturn(proListDisplay);
            List<Product> proList = operaService.displayProducts();

            assertEquals(3, proList.size());

            assertEquals("Turkey breast", proList.get(0).getName());
            assertEquals("Dish", proList.get(0).getType());
            assertEquals(8.99, proList.get(0).getPrice(), 0.01);

            assertEquals("Vegetables", proList.get(1).getName());
            assertEquals("Side", proList.get(1).getType());
            assertEquals(2.99, proList.get(1).getPrice(), 0.01);

            assertEquals("Lemonade", proList.get(2).getName());
            assertEquals("Drink", proList.get(2).getType());
            assertEquals(1.99, proList.get(2).getPrice(), 0.01);
        }

        /**
         * Test display products no product.
         *
         * @author  Wenbo Wang
         * @version 1.0, April 2016
         */
        @Test
        public void testDisplayProductsNoProduct() {

        Mockito.when(productMapper.selectByExample(Mockito.any(ProductExample.class))).t
henReturn(new ArrayList<Product>());
            List<Product> proList = operaService.displayProducts();
            assertEquals(0, proList.size());
        }

        /**
```

```
 * Test create product success.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public void testCreateProductSuccess() {
	int num = 0;
	if (!"".equals(product.getName()) && !"".equals(product.getType())
			&& product.getPrice() != null)
		num = 1;

Mockito.when(productMapper.insert(Mockito.any(Product.class))).thenReturn(num);
	int res = operaService.createProduct(product);
	assertEquals(1, res);
}

/**
 * Test create product no name.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public void testCreateProductNoName() {
	int num = 0;
	product.setName("");
	if (!"".equals(product.getName()) && !"".equals(product.getType())
			&& product.getPrice() != null)
		num = 1;

Mockito.when(productMapper.insert(Mockito.any(Product.class))).thenReturn(num);
	int res = operaService.createProduct(product);
	assertEquals(0, res);
}

/**
 * Test create product no type.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public void testCreateProductNoType() {
	int num = 0;
	product.setType("");
	if (!"".equals(product.getName()) && !"".equals(product.getType())
			&& product.getPrice() != null)
```

```
                num = 1;

Mockito.when(productMapper.insert(Mockito.any(Product.class))).thenReturn(num);
        int res = operaService.createProduct(product);
        assertEquals(0, res);
}

/**
 * Test create product no price.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public void testCreateProductNoPrice() {
        int num = 0;
        product.setPrice(null);
        if (!"".equals(product.getName()) && !"".equals(product.getType())
                        && product.getPrice() != null)
                num = 1;

Mockito.when(productMapper.insert(Mockito.any(Product.class))).thenReturn(num);
        int res = operaService.createProduct(product);
        assertEquals(0, res);
}

/**
 * Test display orders success.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testDisplayOrdersSuccess() {

Mockito.when(orderMapper.selectByExample(Mockito.any())).thenReturn(orderList);
        List<Order> list = operaService.displayOrders();
        assertEquals(1, list.size());
        assertEquals("(305)474-2378", list.get(0).getPhone());
        assertEquals("123 NW 14th St, Miami FL 33100", list.get(0).getShipAddress());
        assertEquals("processing", list.get(0).getStatus());
}

/**
 * Test display orders no order.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
```

```
	 */
	@Test
	public final void testDisplayOrdersNoOrder() {
		Mockito.when(orderMapper.selectByExample(Mockito.any())).thenReturn(new
ArrayList<Order>());
		List<Order> list = operaService.displayOrders();
		assertEquals(0, list.size());
	}
}
```

- ServiceServiceTest

```
package unit.testcase.service;

import static org.junit.Assert.*;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.runners.MockitoJUnitRunner;

import edu.fiu.hmts.dao.hmts_repos.*;
import edu.fiu.hmts.domain.hmts_repos.*;
import edu.fiu.hmts.service.impl.ServiceService;

/**
 * The Class ServiceServiceTest.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@RunWith(MockitoJUnitRunner.class)
public class ServiceServiceTest {

	/** The serv service. */
	@InjectMocks
	ServiceService servService;

	/** The sec question mapper. */
	@Mock
```

```
SecQuestionMapper secQuestionMapper;

/** The product mapper. */
@Mock
ProductMapper productMapper;

/** The sel product mapper. */
@Mock
SelProductMapper selProductMapper;

/** The order mapper. */
@Mock
OrderMapper orderMapper;

/** The order product mapper. */
@Mock
OrderProductMapper orderProductMapper;

/** The payment mapper. */
@Mock
PaymentMapper paymentMapper;

/** The card mapper. */
@Mock
CardMapper cardMapper;

/** The products. */
List<Product> products = new ArrayList<>();

/** The sel products. */
List<SelProduct> selProducts = new ArrayList<>();

/** The sel product. */
SelProduct selProduct = new SelProduct();

/** The order. */
Order order = new Order();

/** The payment. */
Payment payment = new Payment();

/** The ord products. */
List<OrderProduct> ordProducts = new ArrayList<>();

/** The card. */
Card card = new Card();

/** The questions. */
```

```
List<SecQuestion> questions = new ArrayList<>();

/**
 * Sets the up.
 *
 * @throws Exception
 *             the exception
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Before
public void setUp() throws Exception {
        Product product1 = new Product();
        product1.setName("Turkey breast");
        product1.setType("Dish");
        product1.setPrice(8.99);
        products.add(product1);

        Product product2 = new Product();
        product2.setName("Vegetables");
        product2.setType("Side");
        product2.setPrice(2.99);
        products.add(product2);

        Product product3 = new Product();
        product3.setName("Lemonade");
        product3.setType("Drink");
        product3.setPrice(1.99);
        products.add(product3);

        SelProduct selProduct1 = new SelProduct();
        selProduct1.setProductId(1L);
        selProduct1.setQuantity(3);
        selProduct1.setUserId(3L);
        selProducts.add(selProduct1);

        SelProduct selProduct2 = new SelProduct();
        selProduct2.setProductId(2L);
        selProduct2.setQuantity(1);
        selProduct2.setUserId(3L);
        selProducts.add(selProduct2);

        selProduct.setProductId(1L);
        selProduct.setQuantity(5);
        selProduct.setUserId(2L);

        order.setPhone("(305)474-2378");
```

```
        order.setShipAddress("123 NW 14th St, Miami FL 33100");
        order.setStatus("processing");
        order.setOrderId(5L);
        order.setUserId(3L);

        payment.setAmount(8.99);
        payment.setMethod("0");
        payment.setOrderId(5L);
        payment.setPaymentId(6L);

        card.setCardNum("37283937283737283");
        card.setCardOwner("David John");
        DateFormat dateFormat = new SimpleDateFormat("MM/DD/YYYY");
        card.setExpDate(dateFormat.parse("05/01/2020"));
        card.setSecCode("876");
        card.setBillAddress("123 NW 14th St, Miami FL 33100");

        OrderProduct orderProduct = new OrderProduct();
        orderProduct.setAmount(8.99);
        orderProduct.setName("Pizza");
        orderProduct.setPrice(8.99);
        orderProduct.setQuantity(1);
        orderProduct.setType("Side");
        ordProducts.add(orderProduct);

        SecQuestion secQuestion = new SecQuestion();
        secQuestion.setSecQuestionId(1);
        secQuestion.setContent("What's your favorite sport?");
        questions.add(secQuestion);
}

/**
 * Test display menu.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testDisplayMenu() {

Mockito.when(productMapper.selectByExample(Mockito.any())).thenReturn(products);
        List<Product> proList = servService.displayMenu();

        assertEquals(3, proList.size());

        assertEquals("Turkey breast", proList.get(0).getName());
        assertEquals("Dish", proList.get(0).getType());
        assertEquals(8.99, proList.get(0).getPrice(), 0.01);
```

```
        assertEquals("Vegetables", proList.get(1).getName());
        assertEquals("Side", proList.get(1).getType());
        assertEquals(2.99, proList.get(1).getPrice(), 0.01);

        assertEquals("Lemonade", proList.get(2).getName());
        assertEquals("Drink", proList.get(2).getType());
        assertEquals(1.99, proList.get(2).getPrice(), 0.01);
}

/**
 * Test select product success.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testSelectProductSuccess() {
        Mockito.when(selProductMapper.insert(Mockito.any())).thenReturn(1);
        int res = servService.selectProduct(selProduct);
        assertEquals(1, res);
}

/**
 * Test select product no pro id.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testSelectProductNoProId() {
        selProduct.setProductId(null);
        Mockito.when(selProductMapper.insert(Mockito.any())).thenReturn(0);
        int res = servService.selectProduct(selProduct);
        assertEquals(0, res);
}

/**
 * Test select product no qty.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testSelectProductNoQty() {
        selProduct.setProductId(1L);
        selProduct.setQuantity(null);
        Mockito.when(selProductMapper.insert(Mockito.any())).thenReturn(0);
```

```
        int res = servService.selectProduct(selProduct);
        assertEquals(0, res);
}

/**
 * Test select product no user id.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testSelectProductNoUserId() {
        selProduct.setQuantity(5);
        selProduct.setUserId(null);
        Mockito.when(selProductMapper.insert(Mockito.any())).thenReturn(0);
        int res = servService.selectProduct(selProduct);
        assertEquals(0, res);
}

/**
 * Test remove product success.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testRemoveProductSuccess() {

Mockito.when(selProductMapper.deleteByExample(Mockito.any())).thenReturn(1);
        int res = servService.removeProduct(3, 2);

        assertEquals(1, res);
}

/**
 * Test remove product valid user id.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testRemoveProductValidUserId() {

Mockito.when(selProductMapper.deleteByExample(Mockito.any())).thenReturn(0);
        int res = servService.removeProduct(2, 2);

        assertEquals(0, res);
}
```

```
    /**
     * Test remove product invalid user id.
     *
     * @author  Wenbo Wang
     * @version 1.0, April 2016
     */
    @Test
    public final void testRemoveProductInvalidUserId() {

    Mockito.when(selProductMapper.deleteByExample(Mockito.any())).thenReturn(0);
            int res = servService.removeProduct(-1, 2);

            assertEquals(0, res);
    }

    /**
     * Test remove product valid pro id.
     *
     * @author  Wenbo Wang
     * @version 1.0, April 2016
     */
    @Test
    public final void testRemoveProductValidProId() {

    Mockito.when(selProductMapper.deleteByExample(Mockito.any())).thenReturn(0);
            int res = servService.removeProduct(3, 3);

            assertEquals(0, res);
    }

    /**
     * Test display cart success.
     *
     * @author  Wenbo Wang
     * @version 1.0, April 2016
     */
    @Test
    public final void testDisplayCartSuccess() {

    Mockito.when(selProductMapper.selectByExample(Mockito.any())).thenReturn(selProducts);
            List<SelProduct> selProList = servService.displayCart(3L);

            assertEquals(2, selProList.size());

            assertEquals(1L, selProList.get(0).getProductId(), 1);
            assertEquals(3, selProList.get(0).getQuantity(), 1);
```

```
        assertEquals(3L, selProList.get(0).getUserId(), 1);

        assertEquals(2L, selProList.get(1).getProductId(), 1);
        assertEquals(1, selProList.get(1).getQuantity(), 1);
        assertEquals(3L, selProList.get(1).getUserId(), 1);
}

/**
 * Test display cart valid user id.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testDisplayCartValidUserId() {
        List<SelProduct> list = new ArrayList<>();

Mockito.when(selProductMapper.selectByExample(Mockito.any())).thenReturn(list);
        List<SelProduct> selProList = servService.displayCart(2L);

        assertEquals(0, selProList.size());
}

/**
 * Test display cart invalid user id.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testDisplayCartInvalidUserId() {
        List<SelProduct> list = new ArrayList<>();

Mockito.when(selProductMapper.selectByExample(Mockito.any())).thenReturn(list);
        List<SelProduct> selProList = servService.displayCart(-1L);

        assertEquals(0, selProList.size());
}

/**
 * Test place orderby cash success.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testPlaceOrderbyCashSuccess() {
        Mockito.when(orderMapper.insert(Mockito.any())).thenReturn(1);
```

```
        Mockito.when(orderProductMapper.insert(Mockito.any())).thenReturn(1);
        Mockito.when(paymentMapper.insertSelective(Mockito.any())).thenReturn(1);
        int res = servService.placeOrder(order, payment, ordProducts, card);
        assertEquals(1, res);
}

/**
 * Test place orderby cash invalid order.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testPlaceOrderbyCashInvalidOrder() {
        order.setUserId(null);
        Mockito.when(orderMapper.insert(Mockito.any())).thenReturn(0);
        Mockito.when(orderProductMapper.insert(Mockito.any())).thenReturn(1);
        Mockito.when(paymentMapper.insertSelective(Mockito.any())).thenReturn(1);
        int res = servService.placeOrder(order, payment, ordProducts, card);
        assertEquals(0, res);
}

/**
 * Test place orderby cash null order.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testPlaceOrderbyCashNullOrder() {
        Mockito.when(orderMapper.insert(Mockito.any())).thenReturn(1);
        Mockito.when(orderProductMapper.insert(Mockito.any())).thenReturn(1);
        Mockito.when(paymentMapper.insertSelective(Mockito.any())).thenReturn(1);
        int res = servService.placeOrder(null, payment, ordProducts, card);
        assertEquals(0, res);
}

/**
 * Test place orderby cash no ord product.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testPlaceOrderbyCashNoOrdProduct() {
        order.setUserId(3L);
        order.setOrderId(5L);
        List<OrderProduct> ordProList = new ArrayList<>();
```

```
        Mockito.when(orderMapper.insert(Mockito.any())).thenReturn(1);
        Mockito.when(orderProductMapper.insert(Mockito.any())).thenReturn(1);
        Mockito.when(paymentMapper.insertSelective(Mockito.any())).thenReturn(1);
        int res = servService.placeOrder(order, payment, ordProList, card);
        assertEquals(0, res);
}

/**
 * Test place orderby cash invalid ord product.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testPlaceOrderbyCashInvalidOrdProduct() {
        ordProducts.get(0).setName(null);
        Mockito.when(orderMapper.insert(Mockito.any())).thenReturn(1);
        Mockito.when(orderProductMapper.insert(Mockito.any())).thenReturn(0);
        Mockito.when(paymentMapper.insertSelective(Mockito.any())).thenReturn(1);
        int res = servService.placeOrder(order, payment, ordProducts, card);
        assertEquals(0, res);
}

/**
 * Test place orderby cash invalid payment.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testPlaceOrderbyCashInvalidPayment() {
        ordProducts.get(0).setName("Pizza");
        payment.setMethod(null);
        Mockito.when(orderMapper.insert(Mockito.any())).thenReturn(1);
        Mockito.when(orderProductMapper.insert(Mockito.any())).thenReturn(1);
        Mockito.when(paymentMapper.insertSelective(Mockito.any())).thenReturn(0);
        int res = servService.placeOrder(order, payment, ordProducts, card);
        assertEquals(0, res);
}

/**
 * Test place orderby card success.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testPlaceOrderbyCardSuccess() {
```

```
        payment.setMethod("1");
        Mockito.when(orderMapper.insert(Mockito.any())).thenReturn(1);
        Mockito.when(orderProductMapper.insert(Mockito.any())).thenReturn(1);
        Mockito.when(paymentMapper.insertSelective(Mockito.any())).thenReturn(1);
        Mockito.when(cardMapper.insert(Mockito.any())).thenReturn(1);
        int res = servService.placeOrder(order, payment, ordProducts, card);
        assertEquals(1, res);
    }

    /**
     * Test place orderby card invalid card.
     *
     * @author  Wenbo Wang
     * @version 1.0, April 2016
     */
    @Test
    public final void testPlaceOrderbyCardInvalidCard() {
        payment.setMethod("1");
        card.setCardNum(null);
        Mockito.when(orderMapper.insert(Mockito.any())).thenReturn(1);
        Mockito.when(orderProductMapper.insert(Mockito.any())).thenReturn(1);
        Mockito.when(paymentMapper.insertSelective(Mockito.any())).thenReturn(1);
        Mockito.when(cardMapper.insert(Mockito.any())).thenReturn(0);
        int res = servService.placeOrder(order, payment, ordProducts, card);
        assertEquals(0, res);
    }

    /**
     * Test get questions success.
     *
     * @author  Wenbo Wang
     * @version 1.0, April 2016
     */
    @Test
    public final void testGetQuestionsSuccess() {

    Mockito.when(secQuestionMapper.selectByExample(Mockito.any())).thenReturn(questio
ns);
        List<SecQuestion> secQuesList = servService.getQuestions();

        assertEquals(1, secQuesList.size());
        assertEquals(1, secQuesList.get(0).getSecQuestionId(), 1);
        assertEquals("What's your favorite sport?", secQuesList.get(0).getContent());
    }

    /**
     * Test get questions no question.
     *
```

```
         * @author  Wenbo Wang
         * @version 1.0, April 2016
         */
        @Test
        public final void testGetQuestionsNoQuestion() {

        Mockito.when(secQuestionMapper.selectByExample(Mockito.any())).thenReturn(new
ArrayList<SecQuestion>());
                List<SecQuestion> secQuesList = servService.getQuestions();

                assertEquals(0, secQuesList.size());
        }
}
```

11.6.2. Subsystem Test Codes

- **Test Suite**

```
package subsystem;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

import subsystem.testcase.controller.FrontControllerTest;
import subsystem.testcase.controller.OperationControllerTest;
import subsystem.testcase.controller.UserControllerTest;

/**
 * The Class TestSuite.
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@RunWith(Suite.class)
@SuiteClasses({OperationControllerTest.class,
        UserControllerTest.class,
        FrontControllerTest.class,
        OperationControllerTest.class
        MobileInterfaceTest.class})
public class TestSuite {}
```

- **FrontControllerTest**

```
package subsystem.testcase.controller;

import org.junit.Before;
```

```
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.TestExecutionListeners;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.support.DependencyInjectionTestExecutionListener;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.result.MockMvcResultHandlers;
import org.springframework.web.context.WebApplicationContext;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;
import static org.springframework.test.web.servlet.setup.MockMvcBuilders.*;


/**
 * The Class FrontControllerTest.
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@ContextConfiguration(locations = {"file:src/main/webapp/WEB-INF/spring/web-servlet.xml",
                "file:src/main/webapp/WEB-INF/spring/spring-jdbc.xml",
                "file:src/test/resources/testcontext.xml"})
@TestExecutionListeners({DependencyInjectionTestExecutionListener.class})
public class FrontControllerTest {

        /** The wac. */
        @Autowired
        private WebApplicationContext wac;

        /** The mock mvc. */
        MockMvc mockMvc;

        /**
         * Sets the up.
         *
         * @throws Exception
         *             the exception
         */
        @Before
        public void setUp() throws Exception {
                mockMvc = webAppContextSetup(wac).build();
        }
```

```
/**
 * Test dispatcher login.
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testDispatcherLogin() {
	try {
		mockMvc.perform(get("/front.do").param("mode", "dispatcher")
				.param("target", "login")
				.param("userId", "1")
				.param("role", "1")
				.param("firstName", "Administrator")
				.param("func", "1,2,3,"))
			//.andDo(MockMvcResultHandlers.print())
			.andExpect(status().isOk())
			.andExpect(view().name("login"))
			.andExpect(model().attribute("userId", "1"))
			.andExpect(model().attribute("role", "1"))
			.andExpect(model().attribute("firstName", "Administrator"))
			.andExpect(model().attribute("func", "1,2,3,"));
	} catch (Exception e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
}

/**
 * Test dispatcher home.
 * @author Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testDispatcherHome() {
	try {
		mockMvc.perform(get("/front.do").param("mode", "dispatcher")
				.param("target", "home")
				.param("userId", "1")
				.param("role", "1")
				.param("firstName", "Administrator")
				.param("func", "1,2,3,"))
			//.andDo(MockMvcResultHandlers.print())
			.andExpect(status().isOk())
			.andExpect(view().name("home"))
			.andExpect(model().attribute("userId", "1"))
			.andExpect(model().attribute("role", "1"))
			.andExpect(model().attribute("firstName", "Administrator"))
			.andExpect(model().attribute("func", "1,2,3,"));
```

```
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

/**
 * Test dispatcher contact.
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testDispatcherContact() {
        try {
                mockMvc.perform(get("/front.do").param("mode", "dispatcher")
                                .param("target", "contact")
                                .param("userId", "1")
                                .param("role", "1")
                                .param("firstName", "Administrator")
                                .param("func", "1,2,3,"))
                        //.andDo(MockMvcResultHandlers.print())
                        .andExpect(status().isOk())
                        .andExpect(view().name("contact"))
                        .andExpect(model().attribute("userId", "1"))
                        .andExpect(model().attribute("role", "1"))
                        .andExpect(model().attribute("firstName", "Administrator"))
                        .andExpect(model().attribute("func", "1,2,3,"));
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

/**
 * Test dispatcher team.
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testDispatcherTeam() {
        try {
                mockMvc.perform(get("/front.do").param("mode", "dispatcher")
                                .param("target", "team")
                                .param("userId", "1")
                                .param("role", "1")
                                .param("firstName", "Administrator")
                                .param("func", "1,2,3,"))
                        //.andDo(MockMvcResultHandlers.print())
```

```
                        .andExpect(status().isOk())
                        .andExpect(view().name("team"))
                        .andExpect(model().attribute("userId", "1"))
                        .andExpect(model().attribute("role", "1"))
                        .andExpect(model().attribute("firstName", "Administrator"))
                        .andExpect(model().attribute("func", "1,2,3,"));
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

/**
 * Test dispatcher invalid target.
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testDispatcherInvalidTarget() {
        try {
                mockMvc.perform(get("/front.do").param("mode", "dispatcher")
                                .param("target", "WebPage")
                                .param("userId", "1")
                                .param("role", "1")
                                .param("firstName", "Administrator")
                                .param("func", "1,2,3,"))
                        //.andDo(MockMvcResultHandlers.print())
                        .andExpect(status().isOk())
                        .andExpect(view().name("error"));
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

/**
 * Test dispatcher exception.
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testDispatcherException() {
        try {
                mockMvc.perform(get("/front.do").param("mode", "dispatcher")
                                .param("target", (String)null)
                                .param("userId", "1")
                                .param("role", "1")
                                .param("firstName", "Administrator")
```

```
                                .param("func", "1,2,3,"))
                        //.andDo(MockMvcResultHandlers.print())
                        .andExpect(status().isOk())
                        .andExpect(view().name("home"));
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
}
```

- **UserControllerTest**

```
package subsystem.testcase.controller;

import java.util.ArrayList;
import java.util.List;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.TestExecutionListeners;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.support.DependencyInjectionTestExecutionListener;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.web.context.WebApplicationContext;

import static org.mockito.Matchers.any;
import static org.mockito.Mockito.*;
import static org.hamcrest.Matchers.*;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;
import static org.springframework.test.web.servlet.setup.MockMvcBuilders.*;

import edu.fiu.hmts.dao.hmts_repos.FunctionRoleMapper;
import edu.fiu.hmts.dao.hmts_repos.UserMapper;
import edu.fiu.hmts.domain.hmts_repos.FunctionRoleKey;
import edu.fiu.hmts.domain.hmts_repos.User;
import edu.fiu.hmts.service.impl.UserService;

/**
 * The Class UserControllerTest.
 * @author  Wenbo Wang
 * @version 1.0, April 2016
```

```
 */
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@ContextConfiguration(locations = {"file:src/main/webapp/WEB-INF/spring/web-servlet.xml",
                "file:src/main/webapp/WEB-INF/spring/spring-jdbc.xml",
                "file:src/test/resources/testcontext.xml"})
@TestExecutionListeners({DependencyInjectionTestExecutionListener.class})
public class UserControllerTest {

        /** The wac. */
        @Autowired
        private WebApplicationContext wac;

        /** The mock mvc. */
        private MockMvc mockMVC;

        /** The user service. */
        @Autowired
        private UserService userService;

        /** The user mapper. */
        @Autowired
        private UserMapper userMapper;

        /** The function role mapper. */
        @Autowired
        private FunctionRoleMapper functionRoleMapper;

        /** The mob user s. */
        User userNU, webUserS, mobUserS;

        /** The user list f. */
        List<User> webUserList, mobUserList, userListF;

        /** The function role keys2. */
        List<FunctionRoleKey> functionRoleKeys, functionRoleKeys2;

        /**
         * Sets the up.
         *
         * @author  Wenbo Wang
         * @version 1.0, April 2016
         *
         * @throws Exception
         *             the exception
         */
        @Before
        public void setUp() throws Exception {
```

```
        mockMVC = webAppContextSetup(wac).build();

        userNU = new User();
        userNU.setUserId(-1L);

        webUserS = new User();
        webUserS.setUserId(1L);
        webUserS.setUsername("peter@gmail.com");
        webUserS.setPassword("123456");
        webUserS.setFirstName("Peter");
        webUserS.setLastName("John");
        webUserS.setPhone("(305)837-4827");
        webUserS.setRole("2");

        mobUserS = new User();
        mobUserS.setUserId(2L);
        mobUserS.setUsername("david@yahoo.com");
        mobUserS.setPassword("123456");
        mobUserS.setFirstName("David");
        mobUserS.setLastName("Chris");
        mobUserS.setPhone("(305)233-9483");
        mobUserS.setSecQuestionId(1);
        mobUserS.setSecAnswer("Paul");
        mobUserS.setRole("4");

        userListF = new ArrayList<>();
        mobUserList = new ArrayList<>();
        webUserList = new ArrayList<>();
        webUserList.add(webUserS);
        mobUserList.add(mobUserS);

        functionRoleKeys = new ArrayList<>();
        FunctionRoleKey functionRoleKey1 = new FunctionRoleKey();
        functionRoleKey1.setFunctionId(1);
        functionRoleKeys.add(functionRoleKey1);
        FunctionRoleKey functionRoleKey2 = new FunctionRoleKey();
        functionRoleKey2.setFunctionId(2);
        functionRoleKeys.add(functionRoleKey2);

        functionRoleKeys2 = new ArrayList<>();
}

/**
 * Test login web success.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
```

```
	@Test
	public final void testLoginWebSuccess() {
		try {
			doCallRealMethod().when(userService).login(anyString(), anyString(),
anyInt());
			doCallRealMethod().when(userService).rolesControl(any());
			when(userMapper.selectByExample(any())).thenReturn(webUserList);

	when(functionRoleMapper.selectByExample(any())).thenReturn(functionRoleKeys);

			mockMVC.perform(post("/user.do").param("mode", "login")
					.param("username", webUserS.getUsername())
					.param("password", webUserS.getPassword()))
				//MockMvcResultHandlers.andDo(print())
				.andExpect(status().isOk())
				.andExpect(view().name("home"))
				.andExpect(model().attribute("userId", webUserS.getUserId()))
				.andExpect(model().attribute("role", webUserS.getRole()))
				.andExpect(model().attribute("firstName",
webUserS.getFirstName()))
				.andExpect(model().attribute("func", webUserS.getTag()));
		}catch(Exception e){
			e.printStackTrace();
		}
	}

	/**
	 * Test login web failed.
	 *
	 * @author  Wenbo Wang
	 * @version 1.0, April 2016
	 */
	@Test
	public final void testLoginWebFailed() {
		try {
			doCallRealMethod().when(userService).login(anyString(), anyString(),
anyInt());
			doCallRealMethod().when(userService).rolesControl(any());
			when(userMapper.selectByExample(any())).thenReturn(userListF);

	when(functionRoleMapper.selectByExample(any())).thenReturn(functionRoleKeys2);

			mockMVC.perform(post("/user.do").param("mode", "login")
					.param("username", webUserS.getUsername())
					.param("password", webUserS.getPassword()))
				//.andDo(print())
				.andExpect(status().isOk())
				.andExpect(view().name("login"))
```

```
                        .andExpect(model().attribute("username",
webUserS.getUsername()));
                }catch(Exception e){
                        e.printStackTrace();
                }
        }

        /**
         * Test login web exception.
         *
         * @author  Wenbo Wang
         * @version 1.0, April 2016
         */
        @Test
        public final void testLoginWebException() {
                try {
                        doCallRealMethod().when(userService).login(anyString(), anyString(),
anyInt());
                        doCallRealMethod().when(userService).rolesControl(any());
                        when(userMapper.selectByExample(any())).thenReturn(userListF);

        when(functionRoleMapper.selectByExample(any())).thenReturn(functionRoleKeys2);

                        mockMVC.perform(post("/user.do").param("mode", "login")
                                        .param("username", (String)null)
                                        .param("password", webUserS.getPassword()))
                                //.andDo(print())
                                .andExpect(status().isOk())
                                .andExpect(view().name("login"));
                }catch(Exception e){
                        e.printStackTrace();
                }
        }

        /**
         * Test logout.
         *
         * @author  Wenbo Wang
         * @version 1.0, April 2016
         */
        @Test
        public final void testLogout() {
                try {
                        doCallRealMethod().when(userService).logout(anyLong());

                        mockMVC.perform(post("/user.do").param("mode", "logout"))
                                //.andDo(print())
                                .andExpect(status().is3xxRedirection())
```

```
        .andExpect(redirectedUrl("front.do?mode=dispatcher&target=login"))
                        .andExpect(model().attribute("mode", "dispatcher"))
                        .andExpect(model().attribute("target", "login"));
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

/**
 * Test register web success.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testRegisterWebSuccess() {
        try {
                doCallRealMethod().when(userService).displayUsers();
                doCallRealMethod().when(userService).register(anyString(), anyString()
                                , anyString(), anyString(), anyString(), anyString(), anyInt()
                                , anyString(), anyInt());
                when(userMapper.selectByExample(any())).thenReturn(webUserList);
                when(userMapper.insert(any())).thenReturn(1);

                mockMVC.perform(post("/user.do")
                                .param("mode", "register")
                                .param("userId", webUserS.getUserId().toString())
                                .param("firstName", webUserS.getFirstName())
                                .param("role", webUserS.getRole())
                                .param("func", webUserS.getTag())
                                .param("usernameNew", webUserS.getUsername())
                                .param("firstNameNew", webUserS.getFirstName())
                                .param("lastNameNew", webUserS.getLastName())
                                .param("roleNew", "Staff"))
                        //.andDo(MockMvcResultHandlers.print())
                        .andExpect(status().isOk())
                        .andExpect(view().name("users"))
                        .andExpect(model().attribute("userId",
webUserS.getUserId().toString()))
                        .andExpect(model().attribute("firstName",
webUserS.getFirstName()))
                        .andExpect(model().attribute("func", webUserS.getTag()))
                        .andExpect(model().attribute("userList",
containsString("\"firstName\":\"Peter\"")))
                        .andExpect(model().attribute("userList",
containsString("\"lastName\":\"John\"")))
```

```
                                .andExpect(model().attribute("userList",
containsString("\"password\":\"123456\"")))
                                .andExpect(model().attribute("userList",
containsString("\"role\":\"2\"")))
                                .andExpect(model().attribute("userList",
containsString("\"phone\":\"(305)837-4827\"")))
                                .andExpect(model().attribute("userList",
containsString("\"userId\":1")))
                                .andExpect(model().attribute("userList",
containsString("\"username\":\"peter@gmail.com\"")));
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }

        /**
         * Test register web failed.
         *
         * @author  Wenbo Wang
         * @version 1.0, April 2016
         */
        @Test
        public final void testRegisterWebFailed() {
                try {
                        doCallRealMethod().when(userService).displayUsers();
                        doCallRealMethod().when(userService).register(anyString(), anyString()
                                        , anyString(), anyString(), anyString(), anyString(), anyInt()
                                        , anyString(), anyInt());
                        when(userMapper.selectByExample(any())).thenReturn(userListF);
                        when(userMapper.insert(any())).thenReturn(0);

                        mockMVC.perform(post("/user.do")
                                        .param("mode", "register")
                                        .param("userId", webUserS.getUserId().toString())
                                        .param("firstName", webUserS.getFirstName())
                                        .param("role", webUserS.getRole())
                                        .param("func", webUserS.getTag())
                                        .param("usernameNew", webUserS.getUsername())
                                        .param("firstNameNew", webUserS.getFirstName())
                                        .param("lastNameNew", webUserS.getLastName())
                                        .param("roleNew", "Employee"))
                                //.andDo(MockMvcResultHandlers.print())
                                .andExpect(status().isOk())
                                .andExpect(view().name("users"))
                                .andExpect(model().attribute("userId",
webUserS.getUserId().toString()))
```

```
                .andExpect(model().attribute("firstName",
webUserS.getFirstName()))
                .andExpect(model().attribute("func", webUserS.getTag()))
                .andExpect(model().attribute("userList", "[]"));
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    /**
     * Test display users success.
     *
     * @author  Wenbo Wang
     * @version 1.0, April 2016
     */
    @Test
    public final void testDisplayUsersSuccess() {
        try {
            doCallRealMethod().when(userService).displayUsers();
            when(userMapper.selectByExample(any())).thenReturn(webUserList);

            mockMVC.perform(post("/user.do")
                    .param("mode", "displayUsers")
                    .param("userId", webUserS.getUserId().toString())
                    .param("firstName", webUserS.getFirstName())
                    .param("role", webUserS.getRole())
                    .param("func", webUserS.getTag()))
                //.andDo(print())
                .andExpect(status().isOk())
                .andExpect(view().name("users"))
                .andExpect(model().attribute("userId",
webUserS.getUserId().toString()))
                .andExpect(model().attribute("role", webUserS.getRole()))
                .andExpect(model().attribute("firstName",
webUserS.getFirstName()))
                .andExpect(model().attribute("func", webUserS.getTag()))
                .andExpect(model().attribute("userList",
containsString("\"firstName\":\"Peter\"")))
                .andExpect(model().attribute("userList",
containsString("\"lastName\":\"John\"")))
                .andExpect(model().attribute("userList",
containsString("\"password\":\"123456\"")))
                .andExpect(model().attribute("userList",
containsString("\"role\":\"2\"")))
                .andExpect(model().attribute("userList",
containsString("\"phone\":\"(305)837-4827\"")))
```

```
                .andExpect(model().attribute("userList",
containsString("\"userId\":1")))
                .andExpect(model().attribute("userList",
containsString("\"username\":\"peter@gmail.com\"")));
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    /**
     * Test display users no user.
     *
     * @author  Wenbo Wang
     * @version 1.0, April 2016
     */
    @Test
    public final void testDisplayUsersNoUser() {
        try {
            doCallRealMethod().when(userService).displayUsers();
            when(userMapper.selectByExample(any())).thenReturn(userListF);

            mockMVC.perform(post("/user.do")
                    .param("mode", "displayUsers")
                    .param("userId", webUserS.getUserId().toString())
                    .param("firstName", webUserS.getFirstName())
                    .param("role", webUserS.getRole())
                    .param("func", webUserS.getTag()))
                //.andDo(print())
                .andExpect(status().isOk())
                .andExpect(view().name("users"))
                .andExpect(model().attribute("userId",
webUserS.getUserId().toString()))
                .andExpect(model().attribute("role", webUserS.getRole()))
                .andExpect(model().attribute("firstName",
webUserS.getFirstName()))
                .andExpect(model().attribute("func", webUserS.getTag()))
                .andExpect(model().attribute("userList", "[]"));
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

- **OperationControllerTest**

```
package subsystem.testcase.controller;
```

```
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;

import static org.mockito.Mockito.*;
import static org.hamcrest.Matchers.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.TestExecutionListeners;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.context.support.DependencyInjectionTestExecutionListener;
import org.springframework.test.web.servlet.MockMvc;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;
import static org.springframework.test.web.servlet.setup.MockMvcBuilders.*;

import java.util.ArrayList;
import java.util.List;

import org.springframework.web.context.WebApplicationContext;

import edu.fiu.hmts.dao.hmts_repos.OrderMapper;
import edu.fiu.hmts.dao.hmts_repos.ProductMapper;
import edu.fiu.hmts.domain.hmts_repos.Order;
import edu.fiu.hmts.domain.hmts_repos.Product;
import edu.fiu.hmts.service.impl.OperationService;


/**
 * The Class OperationControllerTest.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@ContextConfiguration(locations = {"file:src/main/webapp/WEB-INF/spring/web-servlet.xml",
                    "file:src/main/webapp/WEB-INF/spring/spring-jdbc.xml",
                    "file:src/test/resources/testcontext.xml"})
@TestExecutionListeners({DependencyInjectionTestExecutionListener.class})
public class OperationControllerTest {

        /** The wac. */
        @Autowired
        private WebApplicationContext wac;
```

```
/** The mock mvc. */
private MockMvc mockMVC;

/** The operation service. */
@Autowired
private OperationService operationService;

/** The product mapper. */
@Autowired
private ProductMapper productMapper;

/** The order mapper. */
@Autowired
private OrderMapper orderMapper;

/** The pro list. */
List<Product> proList = new ArrayList<>();

/** The pro list failed. */
List<Product> proListFailed = new ArrayList<>();

/** The order list. */
List<Order> orderList = new ArrayList<>();

/** The product. */
Product product = new Product();

/** The products. */
String products = "";

/** The product failed. */
String productFailed = "";

/**
 * Sets the up.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 *
 * @throws Exception
 *             the exception
 */
@Before
public void setUp() throws Exception {
        mockMVC = webAppContextSetup(wac).build();
```

```
                products = "[{\"brief\":\"Roasted with a flavorful combination of butter and
seasonings.\""
                                + ",\"productId\":1,\"price\":8.99,\"name\":\"Turkey
Breast\",\"type\":\"Dish\"}"
                                + ",{\"brief\":\"Chopped green pepper, onion, pimiento, corn,
green beans, and green peas.\""
                                +
",\"productId\":2,\"price\":2.99,\"name\":\"Vegetables\",\"type\":\"Side\"}]";

                productFailed = "[{\"brief\":\"Chopped green pepper, onion, pimiento, corn, green
beans, and green peas.\""
                                +
",\"productId\":2,\"price\":2.99,\"name\":\"Vegetables\",\"type\":\"Side\"}]";

                Product product1 = new Product();
                product1.setProductId(1L);
                product1.setName("Turkey Breast");
                product1.setType("Dish");
                product1.setPrice(8.99);
                product1.setBrief("Roasted with a flavorful combination of butter and
seasonings.");
                proList.add(product1);

                Product product2 = new Product();
                product2.setProductId(2L);
                product2.setName("Vegetables");
                product2.setType("Side");
                product2.setPrice(2.99);
                product2.setBrief("Chopped green pepper, onion, pimiento, corn, green beans,
and green peas.");
                proList.add(product2);

                proListFailed.add(product2);

                product.setName("Pizza");
                product.setType("Side");
                product.setPrice(10.99);
                product.setBrief("Pork, Onions and Cheese.");

                Order order = new Order();
                order.setPhone("(305)474-2378");
                order.setShipAddress("123 NW 14th St, Miami FL 33100");
                order.setStatus("processing");
                order.setOrderId(5L);
                order.setUserId(3L);
                orderList.add(order);
        }
```

```
	/**
	 * Test display products success.
	 *
	 * @author  Wenbo Wang
	 * @version 1.0, April 2016
	 */
	@Test
	public final void testDisplayProductsSuccess() {
		try {
			doCallRealMethod().when(operationService).displayProducts();
			when(productMapper.selectByExample(any())).thenReturn(proList);

			mockMVC.perform(post("/operation.do")
					.param("mode", "displayProducts")
					.param("userId", "1")
					.param("firstName", "Administrator")
					.param("role", "1")
					.param("func", "1,2,3,"))
				//.andDo(print())
				.andExpect(status().isOk())
				.andExpect(model().attribute("userId", "1"))
				.andExpect(model().attribute("firstName", "Administrator"))
				.andExpect(model().attribute("func", "1,2,3,"))
				.andExpect(model().attribute("proList", products));
		} catch (Exception e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	/**
	 * Test display products no product.
	 *
	 * @author  Wenbo Wang
	 * @version 1.0, April 2016
	 */
	@Test
	public final void testDisplayProductsNoProduct() {
		try {
			doCallRealMethod().when(operationService).displayProducts();
			when(productMapper.selectByExample(any())).thenReturn(new
ArrayList<Product>());

			mockMVC.perform(post("/operation.do")
					.param("mode", "displayProducts")
					.param("userId", "1")
					.param("firstName", "Administrator")
					.param("role", "1")
```

```
                        .param("func", "1,2,3,"))
                //.andDo(print())
                .andExpect(status().isOk())
                .andExpect(model().attribute("userId", "1"))
                .andExpect(model().attribute("firstName", "Administrator"))
                .andExpect(model().attribute("func", "1,2,3,"))
                .andExpect(model().attribute("proList", "[]"));
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

/**
 * Test create product success.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testCreateProductSuccess() {
        try {
                doCallRealMethod().when(operationService).displayProducts();
                doCallRealMethod().when(operationService).createProduct(any());
                when(productMapper.selectByExample(any())).thenReturn(proList);
                when(productMapper.insert(any())).thenReturn(1);

                mockMVC.perform(post("/operation.do")
                                .param("mode", "createProduct")
                                .param("userId", "1")
                                .param("firstName", "Administrator")
                                .param("role", "1")
                                .param("func", "1,2,3,")
                                .param("name", product.getName())
                                .param("type", product.getType())
                                .param("price", product.getPrice().toString())
                                .param("brief", product.getBrief()))
                        //.andDo(MockMvcResultHandlers.print())
                        .andExpect(status().isOk())
                        .andExpect(model().attribute("userId", "1"))
                        .andExpect(model().attribute("firstName", "Administrator"))
                        .andExpect(model().attribute("func", "1,2,3,"))
                        .andExpect(model().attribute("proList", products));
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}
```

```
/**
 * Test create product no name.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testCreateProductNoName() {
	try {
		doCallRealMethod().when(operationService).displayProducts();
		doCallRealMethod().when(operationService).createProduct(any());
		when(productMapper.selectByExample(any())).thenReturn(proListFailed);
		when(productMapper.insert(any())).thenReturn(1);

		mockMVC.perform(post("/operation.do")
					.param("mode", "createProduct")
					.param("userId", "1")
					.param("firstName", "Administrator")
					.param("role", "1")
					.param("func", "1,2,3,")
					.param("name", (String)null)
					.param("type", product.getType())
					.param("price", product.getPrice().toString())
					.param("brief", product.getBrief()))
				//.andDo(MockMvcResultHandlers.print())
				.andExpect(status().isOk())
				.andExpect(model().attribute("userId", "1"))
				.andExpect(model().attribute("firstName", "Administrator"))
				.andExpect(model().attribute("func", "1,2,3,"))
				.andExpect(model().attribute("proList", productFailed));
	} catch (Exception e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
}

/**
 * Test create product no type.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testCreateProductNoType() {
	try {
		doCallRealMethod().when(operationService).displayProducts();
		doCallRealMethod().when(operationService).createProduct(any());
```

```
                when(productMapper.selectByExample(any())).thenReturn(proListFailed);
                when(productMapper.insert(any())).thenReturn(1);

                mockMVC.perform(post("/operation.do")
                                .param("mode", "createProduct")
                                .param("userId", "1")
                                .param("firstName", "Administrator")
                                .param("role", "1")
                                .param("func", "1,2,3,")
                                .param("name", product.getName())
                                .param("type", (String)null)
                                .param("price", product.getPrice().toString())
                                .param("brief", product.getBrief()))
                        //.andDo(MockMvcResultHandlers.print())
                        .andExpect(status().isOk())
                        .andExpect(model().attribute("userId", "1"))
                        .andExpect(model().attribute("firstName", "Administrator"))
                        .andExpect(model().attribute("func", "1,2,3,"))
                        .andExpect(model().attribute("proList", productFailed));
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

/**
 * Test create product no price.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Test
public final void testCreateProductNoPrice() {
        try {
                doCallRealMethod().when(operationService).displayProducts();
                doCallRealMethod().when(operationService).createProduct(any());
                when(productMapper.selectByExample(any())).thenReturn(proListFailed);
                when(productMapper.insert(any())).thenReturn(1);

                mockMVC.perform(post("/operation.do")
                                .param("mode", "createProduct")
                                .param("userId", "1")
                                .param("firstName", "Administrator")
                                .param("role", "1")
                                .param("func", "1,2,3,")
                                .param("name", product.getName())
                                .param("type", product.getType())
                                .param("price", (String)null)
```

```
                                .param("brief", product.getBrief()))
                        //.andDo(MockMvcResultHandlers.print())
                        .andExpect(status().isOk())
                        .andExpect(model().size(0));
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

        /**
         * Test display orders success.
         *
         * @author  Wenbo Wang
         * @version 1.0, April 2016
         */
        @Test
        public final void testDisplayOrdersSuccess() {
            try {
                doCallRealMethod().when(operationService).displayOrders();
                when(orderMapper.selectByExample(any())).thenReturn(orderList);

                mockMVC.perform(post("/operation.do")
                                .param("mode", "displayOrders")
                                .param("userId", "1")
                                .param("firstName", "Administrator")
                                .param("role", "1")
                                .param("func", "1,2,3,"))
                        //.andDo(MockMvcResultHandlers.print())
                        .andExpect(status().isOk())
                        .andExpect(model().attribute("userId", "1"))
                        .andExpect(model().attribute("firstName", "Administrator"))
                        .andExpect(model().attribute("func", "1,2,3,"))
                        .andExpect(model().attribute("orderList",
containsString("\"orderId\":5")))
                        .andExpect(model().attribute("orderList",
containsString("\"phone\":\"(305)474-2378\"")))
                        .andExpect(model().attribute("orderList",
containsString("\"userId\":3")))
                        .andExpect(model().attribute("orderList",
containsString("\"status\":\"processing\"")))
                        .andExpect(model().attribute("orderList",
containsString("\"shipAddress\":\"123 NW 14th St, Miami FL 33100\"")));
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
```

```
	/**
	 * Test display orders no order.
	 *
	 * @author  Wenbo Wang
	 * @version 1.0, April 2016
	 */
	@Test
	public final void testDisplayOrdersNoOrder() {
		try {
			doCallRealMethod().when(operationService).displayOrders();
			when(orderMapper.selectByExample(any())).thenReturn(new
ArrayList<Order>());

			mockMVC.perform(post("/operation.do")
					.param("mode", "displayOrders")
					.param("userId", "1")
					.param("firstName", "Administrator")
					.param("role", "1")
					.param("func", "1,2,3,"))
				//.andDo(print())
				.andExpect(status().isOk())
				.andExpect(model().attribute("userId", "1"))
				.andExpect(model().attribute("firstName", "Administrator"))
				.andExpect(model().attribute("func", "1,2,3,"))
				.andExpect(model().attribute("orderList", "[]"));
		} catch (Exception e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

}
```

- **MobileInterfaceTest**

```
package subsystem.testcase.controller;

import static org.junit.Assert.*;
import static org.mockito.Matchers.any;
import static org.mockito.Mockito.doCallRealMethod;
import static org.mockito.Mockito.when;

import java.util.ArrayList;
import java.util.List;

import javax.ws.rs.client.Entity;
import javax.ws.rs.core.Application;
```

```
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;

import static org.hamcrest.Matchers.*;

import org.glassfish.jersey.server.ResourceConfig;
import org.glassfish.jersey.test.JerseyTest;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.MockitoAnnotations;

import edu.fiu.hmts.dao.hmts_repos.OrderMapper;
import edu.fiu.hmts.dao.hmts_repos.ProductMapper;
import edu.fiu.hmts.dao.hmts_repos.SelProductMapper;
import edu.fiu.hmts.dao.hmts_repos.UserMapper;
import edu.fiu.hmts.domain.hmts_repos.Product;
import edu.fiu.hmts.domain.hmts_repos.SelProduct;
import edu.fiu.hmts.domain.hmts_repos.User;
import edu.fiu.hmts.service.IServiceService;
import edu.fiu.hmts.service.impl.ServiceService;
import edu.fiu.hmts.service.impl.UserService;
import edu.fiu.hmts.web.service.MobileInterface;
import net.minidev.json.JSONObject;

/**
 * The Class MobileInterfaceTest.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
public class MobileInterfaceTest extends JerseyTest {

	/** The serv service. */
	private ServiceService servService;

	/** The user service. */
	@Mock
	private UserService userService;

	/** The user mapper. */
	@Mock
	private UserMapper userMapper;

	/** The product mapper. */
	@Mock
```

```
private ProductMapper productMapper;

/** The order mapper. */
@Mock
private OrderMapper orderMapper;

/** The sel product mapper. */
@Mock
private SelProductMapper selProductMapper;

/** The mob user s. */
User userNU, mobUserS;

/** The user list f. */
List<User> mobUserList, userListF;

/** The pro list. */
List<Product> proList = new ArrayList<>();

/** The user json. */
JSONObject userJson = new JSONObject();

/**
 * Configure.
 *
 * @return the application
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@Override
protected Application configure() {
	return new ResourceConfig(MobileInterface.class)
				.property("contextConfigLocation", "classpath:testContext.xml");
}

/**
 * Sets the up child.
 */
@Before
public void setUpChild(){
	MockitoAnnotations.initMocks(this);

	servService = Mockito.mock(ServiceService.class);

	mobUserS = new User();
	mobUserS.setUserId(2L);
	mobUserS.setUsername("david@yahoo.com");
```

```
        mobUserS.setPassword("123456");
        mobUserS.setFirstName("David");
        mobUserS.setLastName("Chris");
        mobUserS.setPhone("(305)233-9483");
        mobUserS.setSecQuestionId(1);
        mobUserS.setSecAnswer("Paul");
        mobUserS.setRole("4");

        userJson.put("userId", mobUserS.getUserId());
        userJson.put("username", mobUserS.getUsername());
        userJson.put("password", mobUserS.getPassword());
        userJson.put("firstName", mobUserS.getFirstName());
        userJson.put("lastName", mobUserS.getLastName());
        userJson.put("phone", mobUserS.getPhone());
        userJson.put("secQuestionId", mobUserS.getSecQuestionId());
        userJson.put("secAnswer", mobUserS.getSecAnswer());
        userJson.put("role", mobUserS.getRole());

        mobUserList = new ArrayList<>();
        mobUserList.add(mobUserS);

        Product product1 = new Product();
        product1.setName("Turkey breast");
        product1.setType("Dish");
        product1.setPrice(8.99);
        proList.add(product1);

        Product product2 = new Product();
        product2.setName("Vegetables");
        product2.setType("Side");
        product2.setPrice(2.99);
        proList.add(product2);

        Product product3 = new Product();
        product3.setName("Lemonade");
        product3.setType("Drink");
        product3.setPrice(1.99);
        proList.add(product3);
}

/**
 * Tear down child.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
@After
public void tearDownChild() { }
```

```
	/**
	 * Test register.
	 *
	 * @author  Wenbo Wang
	 * @version 1.0, April 2016
	 */
	@Test
	public final void testRegister() {
		try {
			doCallRealMethod().when(userService).register(any(), any(), any(), any(),
any(), any(), any(), any(), any());
			when(userMapper.insert(any())).thenReturn(1);

			Entity<JSONObject> user = Entity.entity(userJson,
MediaType.APPLICATION_JSON);

			Response response = target("/service/register").request().post(user);
			assertEquals(user, response.getEntity());
		}catch(Exception e){
			e.printStackTrace();
		}
	}

	/**
	 * Test login mob success.
	 *
	 * @author  Wenbo Wang
	 * @version 1.0, April 2016
	 */
	@Test
	public final void testLoginMobSuccess() {
		try {
			doCallRealMethod().when(userService).login(any(), any(), any());
			when(userMapper.selectByExample(any())).thenReturn(mobUserList);

			Entity<JSONObject> user = Entity.entity(userJson,
MediaType.APPLICATION_JSON);

			Response response = target("/service/login").request().post(user);
			assertEquals(user, response.getEntity());
		}catch(Exception e){
			e.printStackTrace();
		}
	}

	/**
	 * Test login mob failed.
```

```
	 *
	 * @author  Wenbo Wang
	 * @version 1.0, April 2016
	 */
	@Test
	public final void testLoginMobFailed() {
		try {
			doCallRealMethod().when(userService).login(any(), any(), any());
			when(userMapper.selectByExample(any())).thenReturn(mobUserList);

			Entity<JSONObject> user = Entity.entity(userJson,
MediaType.APPLICATION_JSON);

			Response response = target("/service/login").request().post(user);
			assertEquals(null, response.getEntity());
		}catch(Exception e){
			e.printStackTrace();
		}
	}

	/**
	 * Test logout.
	 *
	 * @author  Wenbo Wang
	 * @version 1.0, April 2016
	 */
	@Test
	public final void testLogout() {
		try {
			doCallRealMethod().when(userService).logout(any());

			Entity<JSONObject> user = Entity.entity(userJson,
MediaType.APPLICATION_JSON);

			Response response = target("/service/logout").request().post(user);
			assertEquals(true, response.getEntity());
		}catch(Exception e){
			e.printStackTrace();
		}
	}

	/**
	 * Test display menu.
	 *
	 * @author  Wenbo Wang
	 * @version 1.0, April 2016
	 */
	@Test
```

```
	public final void testDisplayMenu() {
		try {
			doCallRealMethod().when(servService).displayMenu();
			when(productMapper.selectByExample(any())).thenReturn(proList);
			//when(servService.displayMenu()).thenReturn(proList);

			String response =
target("/service/displaymenu").request().get(String.class);
		assertEquals("{\"result\":\"successful\",\"count\":0,\"data\":[]}", response);
		}catch(Exception e){
			e.printStackTrace();
		}
	}

	/**
	 * Test select product.
	 *
	 * @author  Wenbo Wang
	 * @version 1.0, April 2016
	 */
	@Test
	public final void testSelectProduct() {
		try {
			doCallRealMethod().when(servService).selectProduct(any());
			when(productMapper.insert(any())).thenReturn(1);

			Entity<Product> product = Entity.entity(proList.get(0),
MediaType.APPLICATION_JSON);

			Response response =
target("/service/selectproect").request().post(product);
			assertEquals(1, response.getEntity());
		}catch(Exception e){
			e.printStackTrace();
		}
	}

	/**
	 * Test remove product.
	 *
	 * @author  Wenbo Wang
	 * @version 1.0, April 2016
	 */
	@Test
	public final void testRemoveProduct() {
		try {
			doCallRealMethod().when(servService).removeProduct(any(), any());
			when(productMapper.deleteByExample(any())).thenReturn(1);
```

```
                Entity<Product> product = Entity.entity(proList.get(0),
MediaType.APPLICATION_JSON);

                Response response =
target("/service/removeproect").request().post(product);
                assertEquals(1, response.getEntity());
            }catch(Exception e){
                e.printStackTrace();
            }
        }

        /**
         * Test display cart.
         *
         * @author  Wenbo Wang
         * @version 1.0, April 2016
         */
        @Test
        public final void testDisplayCart() {
            try {
                doCallRealMethod().when(servService).displayCart(any());
                when(selProductMapper.selectByExample(any())).thenReturn(new
ArrayList<SelProduct>());

                Entity<Product> product = Entity.entity(proList.get(0),
MediaType.APPLICATION_JSON);

                Response response = target("/service/displaycart").request().get();
                assertEquals(1, response.getEntity());
            }catch(Exception e){
                e.printStackTrace();
            }
        }
}
```

11.6.3. System Test Codes

- **Test Suite**

```
package edu.fiu.hmts_cu;

import junit.framework.Test;
import junit.framework.TestSuite;

/**
 * Test all system functions.
```

```
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
public class AllSystemTests {
   public static Test suite() {
      TestSuite suite = new TestSuite(AllSystemTests.class.getName());
      suite.addTest(TestSuite.createTest(testLogin_S1.class, "testRun"));
      suite.addTest(TestSuite.createTest(testLogin_S2.class, "testRun"));
      suite.addTest(TestSuite.createTest(testLogin_R1.class, "testRun"));

      suite.addTest(TestSuite.createTest(testLogout_S1.class, "testRun"));
      suite.addTest(TestSuite.createTest(testLogout_S2.class, "testRun"));

      suite.addTest(TestSuite.createTest(testRegister_S1.class, "testRun"));
      suite.addTest(TestSuite.createTest(testRegister_S2.class, "testRun"));
      suite.addTest(TestSuite.createTest(testRegister_R1.class, "testRun"));

      suite.addTest(TestSuite.createTest(testSelectProduct_S1.class, "testRun"));
      suite.addTest(TestSuite.createTest(testSelectProduct_S2.class, "testRun"));
      suite.addTest(TestSuite.createTest(testSelectProduct_R1.class, "testRun"));

      suite.addTest(TestSuite.createTest(testRemoveProduct_S1.class, "testRun"));
      suite.addTest(TestSuite.createTest(testRemoveProduct_S2.class, "testRun"));
      suite.addTest(TestSuite.createTest(testRemoveProduct_R1.class, "testRun"));

      suite.addTest(TestSuite.createTest(testViewCart_S1.class, "testRun"));
      suite.addTest(TestSuite.createTest(testViewCart_S2.class, "testRun"));
      suite.addTest(TestSuite.createTest(testViewCart_R1.class, "testRun"));

      suite.addTest(TestSuite.createTest(testFillOutOrderInfo_Cash_S1.class, "testRun"));
      suite.addTest(TestSuite.createTest(testFillOutOrderInfo_Cash_S2.class, "testRun"));
      suite.addTest(TestSuite.createTest(testFillOutOrderInfo_Cash_R1.class, "testRun"));

      suite.addTest(TestSuite.createTest(testFillOutOrderInfo_Card_S1.class, "testRun"));
      suite.addTest(TestSuite.createTest(testFillOutOrderInfo_Card_S2.class, "testRun"));
      suite.addTest(TestSuite.createTest(testFillOutOrderInfo_Card_R1.class, "testRun"));

      suite.addTest(TestSuite.createTest(testPopAddressAlert_S1.class, "testRun"));
      suite.addTest(TestSuite.createTest(testPopAddressAlert_S2.class, "testRun"));
      suite.addTest(TestSuite.createTest(testPopAddressAlert_R1.class, "testRun"));

      return suite;
   }
}
```

- **LoginTest**

```
package edu.fiu.hmts_cu;

import edu.fiu.hmts_cu.activity.MenuActivity;
import edu.fiu.hmts_cu.activity.SplashActivity;
import com.robotium.solo.*;
import android.test.ActivityInstrumentationTestCase2;

/**
 * Test Login.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
public class testLogin_S1 extends ActivityInstrumentationTestCase2<SplashActivity> {
  private Solo solo;

  public testLogin_S1() {
    super(SplashActivity.class);
  }

  public void setUp() throws Exception {
      super.setUp();
    solo = new Solo(getInstrumentation());
    getActivity();
  }

     @Override
     public void tearDown() throws Exception {
      solo.finishOpenedActivities();
      super.tearDown();
  }

  public void testRun() {
      //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
    solo.waitForActivity(SplashActivity.class, 2000);
      //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
    assertTrue("edu.fiu.hmts_cu.activity.LoginActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class));
      //Click on Empty Text View
```

```
    solo.clickOnView(solo.getView(R.id.usernm));
     //Enter the text: 'bob@fiu.edu'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
    solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "bob@fiu.edu");
     //Click on Empty Text View
    solo.clickOnView(solo.getView(R.id.passwd));
     //Enter the text: 'bb1234'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
    solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "bb1234");
     //Click on Log In
    solo.clickOnView(solo.getView(R.id.login));
    assertTrue("edu.fiu.hmts_cu.activity.MenuActivity is not found!",
solo.waitForActivity(MenuActivity.class));
    solo.assertCurrentActivity("MenuActivity", MenuActivity.class);
    assertTrue(solo.searchText("Turkey Breast"));
    assertTrue(solo.searchText("8.99"));
    assertTrue(solo.searchText("Vegetables"));
    assertTrue(solo.searchText("2.99"));
    assertTrue(solo.searchText("Lemonade"));
    assertTrue(solo.searchText("1.99"));
  }
}

public void testRun() {
     //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
  solo.waitForActivity(SplashActivity.class, 2000);
     //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
  assertTrue("edu.fiu.hmts_cu.activity.LoginActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class));
     //Enter the text: 'jack@fiu.edu'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
  solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "jack@fiu.edu");
     //Enter the text: 'jk1234'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
  solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "jk1234");
     //Click on Log In
  solo.clickOnView(solo.getView(R.id.login));
     assertTrue("edu.fiu.hmts_cu.activity.MenuActivity is not found!",
solo.waitForActivity(MenuActivity.class));
     solo.assertCurrentActivity("MenuActivity", MenuActivity.class);
```

```
    assertTrue(solo.searchText("Turkey Breast"));
        assertTrue(solo.searchText("8.99"));
        assertTrue(solo.searchText("Vegetables"));
        assertTrue(solo.searchText("2.99"));
        assertTrue(solo.searchText("Lemonade"));
        assertTrue(solo.searchText("1.99"));
}

public void testRun() {
        //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
    solo.waitForActivity(SplashActivity.class, 2000);
        //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
    assertTrue("edu.fiu.hmts_cu.activity.LoginActivity is not found!",
solo.waitForActivity(LoginActivity.class));
        //Enter the text: 'jack@fiu.edu'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
    solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "john@fiu.edu");
        //Enter the text: 'jk1234'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
    solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "jn1234");
        //Click on Log In
    solo.clickOnView(solo.getView(R.id.login));
        assertFalse("edu.fiu.hmts_cu.activity.MenuActivity is found!",
solo.waitForActivity(MenuActivity.class));
        solo.assertCurrentActivity("LoginActivity", LoginActivity.class);
    assertTrue(solo.searchText("Happy Meal"));
}
```

- **LogoutTest**

```
package edu.fiu.hmts_cu;

import android.test.ActivityInstrumentationTestCase2;

import com.robotium.solo.Solo;

import edu.fiu.hmts_cu.activity.LoginActivity;
import edu.fiu.hmts_cu.activity.MenuActivity;
import edu.fiu.hmts_cu.activity.SplashActivity;

/**
 * Test Logout.
 *
```

```
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
public class testLogout_S1 extends ActivityInstrumentationTestCase2<SplashActivity> {
  private Solo solo;

  public testLogout_S1() {
    super(SplashActivity.class);
  }

  public void setUp() throws Exception {
      super.setUp();
    solo = new Solo(getInstrumentation());
    getActivity();
  }

     @Override
     public void tearDown() throws Exception {
      solo.finishOpenedActivities();
      super.tearDown();
  }

  public void testRun() {
      //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
    solo.waitForActivity(SplashActivity.class, 2000);
      //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
    solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class);
      //Click on Empty Text View
    solo.clickOnView(solo.getView(R.id.usernm));
      //Enter the text: 'bob@fiu.edu'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
    solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "bob@fiu.edu");
      //Click on Empty Text View
    solo.clickOnView(solo.getView(R.id.passwd));
      //Enter the text: 'bb1234'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
    solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "bb1234");
      //Click on Log In
    solo.clickOnView(solo.getView(R.id.login));
    //Click on Empty Text View
    solo.clickOnView(solo.getView(edu.fiu.hmts_cu.R.id.header_left_btn));
    assertTrue("edu.fiu.hmts_cu.activity.LoginActivity is not found!",
solo.waitForActivity(LoginActivity.class));
    solo.assertCurrentActivity("LoginActivity", LoginActivity.class);
    assertTrue(solo.searchText("Happy Meal"));
  }
}
```

```
public void testRun() {
      //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
   solo.waitForActivity(SplashActivity.class, 2000);
      //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
   solo.waitForActivity(LoginActivity.class);
      //Click on Empty Text View
   solo.clickOnView(solo.getView(R.id.usernm));
      //Enter the text: 'bob@fiu.edu'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
   solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "jack@fiu.edu");
      //Click on Empty Text View
   solo.clickOnView(solo.getView(R.id.passwd));
      //Enter the text: 'bb1234'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
   solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "jk1234");
      //Click on Log In
   solo.clickOnView(solo.getView(R.id.login));
   //Click on Empty Text View
   solo.clickOnView(solo.getView(R.id.header_left_btn));
   assertTrue("edu.fiu.hmts_cu.activity.LoginActivity is not found!",
solo.waitForActivity(LoginActivity.class));
   solo.assertCurrentActivity("LoginActivity", LoginActivity.class);
   assertTrue(solo.searchText("Happy Meal"));
}
```

- **RegisterTest**

```
package edu.fiu.hmts_cu;

import android.test.ActivityInstrumentationTestCase2;

import com.robotium.solo.Solo;

import edu.fiu.hmts_cu.activity.MenuActivity;
import edu.fiu.hmts_cu.activity.SplashActivity;

/**
 * Test Register.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
public class testRegister_S1 extends ActivityInstrumentationTestCase2<SplashActivity> {
   private Solo solo;

   public testRegister_S1() {
      super(SplashActivity.class);
   }
```

```
  public void setUp() throws Exception {
      super.setUp();
    solo = new Solo(getInstrumentation());
    getActivity();
  }

     @Override
     public void tearDown() throws Exception {
      solo.finishOpenedActivities();
      super.tearDown();
  }

  public void testRun() {
      //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
    solo.waitForActivity(SplashActivity.class, 2000);
      //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
    solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class);
    //Click on Sign up
    solo.clickOnView(solo.getView(edu.fiu.hmts_cu.R.id.newuser));
    //Wait for activity: 'edu.fiu.hmts_cu.activity.RegisterActivity'
    assertTrue("edu.fiu.hmts_cu.activity.RegisterActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.RegisterActivity.class));
    //Enter the text: 'jackson@gmail.com'
    solo.clearEditText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.email));
    solo.enterText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.email),
"jackson@gmail.com");
    //Enter the text: 'jn1234'
    solo.clearEditText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.passwd));
    solo.enterText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.passwd),
"jn1234");
    //Enter the text: 'Jackson'
    solo.clearEditText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.firstnm));
    solo.enterText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.firstnm),
"Jackson");
    //Enter the text: 'Bill'
    solo.clearEditText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.lastnm));
    solo.enterText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.lastnm),
"Bill");
    //Enter the text: '(305)473-8737'
    solo.clearEditText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.phone));
    solo.enterText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.phone),
"(305)473-8737");
    solo.clickOnView(solo.getView(edu.fiu.hmts_cu.R.id.question));
    //Click on What's your favorite sport?
    solo.clickOnView(solo.getView(android.R.id.text1, 2));
    //Click on Empty Text View
    solo.clickOnView(solo.getView(edu.fiu.hmts_cu.R.id.answer));
```

```
    //Enter the text: 'Running'
    solo.clearEditText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.answer));
    solo.enterText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.answer),
"Running");
    //Click on Create account
    solo.clickOnView(solo.getView(edu.fiu.hmts_cu.R.id.signup));
    //Wait for activity: 'edu.fiu.hmts_cu.activity.MenuActivity'
    assertTrue("edu.fiu.hmts_cu.activity.MenuActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.MenuActivity.class));
  }
}


public void testRun() {
    //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
  solo.waitForActivity(SplashActivity.class, 2000);
    //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
  solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class);
  //Click on Sign up
  solo.clickOnView(solo.getView(R.id.newuser));
  //Wait for activity: 'edu.fiu.hmts_cu.activity.RegisterActivity'
  assertTrue("edu.fiu.hmts_cu.activity.RegisterActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.RegisterActivity.class));
  //Enter the text: 'jackson@gmail.com'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.email));
  solo.enterText((android.widget.EditText) solo.getView(R.id.email), "ackerley@gmail.com");
  //Enter the text: 'jn1234'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
  solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "ay1234");
  //Enter the text: 'Jackson'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.firstnm));
  solo.enterText((android.widget.EditText) solo.getView(R.id.firstnm), "Ackerley");
  //Enter the text: 'Bill'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.lastnm));
  solo.enterText((android.widget.EditText) solo.getView(R.id.lastnm), "John");
  //Enter the text: '(305)473-8737'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.phone));
  solo.enterText((android.widget.EditText) solo.getView(R.id.phone), "(305)876-2255");
  solo.clickOnView(solo.getView(R.id.question));
  //Click on Where were you born?
  solo.clickOnView(solo.getView(android.R.id.text1, 1));
  //Click on Empty Text View
  solo.clickOnView(solo.getView(R.id.answer));
  //Enter the text: 'Running'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.answer));
  solo.enterText((android.widget.EditText) solo.getView(R.id.answer), "Miami");
  //Click on Create account
  solo.clickOnView(solo.getView(R.id.signup));
  //Wait for activity: 'edu.fiu.hmts_cu.activity.MenuActivity'
```

```
  assertTrue("edu.fiu.hmts_cu.activity.MenuActivity is not found!",
solo.waitForActivity(MenuActivity.class));
}

public void testRun() {
      //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
  solo.waitForActivity(SplashActivity.class, 2000);
      //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
  solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class);
  //Click on Sign up
  solo.clickOnView(solo.getView(R.id.newuser));
  //Wait for activity: 'edu.fiu.hmts_cu.activity.RegisterActivity'
  assertTrue("edu.fiu.hmts_cu.activity.RegisterActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.RegisterActivity.class));
  //Enter the text: 'jn1234'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
  solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "jn1234");
  //Enter the text: 'Jackson'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.firstnm));
  solo.enterText((android.widget.EditText) solo.getView(R.id.firstnm), "Jackson");
  //Enter the text: 'Bill'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.lastnm));
  solo.enterText((android.widget.EditText) solo.getView(R.id.lastnm), "Bill");
  //Click on Create account
  solo.clickOnView(solo.getView(R.id.signup));
  //Wait for activity: 'edu.fiu.hmts_cu.activity.MenuActivity'
  assertTrue("Text is not found!", solo.searchText("Sorry! Registration information is
incomplete."));
}
```

- **SelectProductTest**

```
package edu.fiu.hmts_cu;

import edu.fiu.hmts_cu.activity.SplashActivity;
import com.robotium.solo.*;
import android.test.ActivityInstrumentationTestCase2;
import android.widget.TextView;

/**
 * Test SelectProduct.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
public class testSelectProduct_S1 extends ActivityInstrumentationTestCase2<SplashActivity> {
  private Solo solo;
```

```
  public testSelectProduct_S1() {
    super(SplashActivity.class);
  }

  public void setUp() throws Exception {
      super.setUp();
    solo = new Solo(getInstrumentation());
    getActivity();
  }

     @Override
     public void tearDown() throws Exception {
      solo.finishOpenedActivities();
      super.tearDown();
  }

  public void testRun() {
      //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
    solo.waitForActivity(SplashActivity.class, 2000);
      //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
    solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class);
      //Enter the text: 'jack@fiu.edu'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
    solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "jack@fiu.edu");
      //Enter the text: 'jk1234'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
    solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "jk1234");
      //Click on Log In
    solo.clickOnView(solo.getView(R.id.login));
      //Wait for activity: 'edu.fiu.hmts_cu.activity.MenuActivity'
    assertTrue("edu.fiu.hmts_cu.activity.MenuActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.MenuActivity.class));
      //Click on Empty Text View
    solo.clickOnView(solo.getView(R.id.prodadd));
    assertEquals("1", ((TextView)solo.getView(R.id.prodquatity)).getText());
  }
}


public void testRun() {
      //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
  solo.waitForActivity(SplashActivity.class, 2000);
      //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
  assertTrue("edu.fiu.hmts_cu.activity.LoginActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class));
      //Enter the text: 'jack@fiu.edu'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
  solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "bob@fiu.edu");
```

```
        //Enter the text: 'jk1234'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
    solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "bb1234");
        //Click on Log In
    solo.clickOnView(solo.getView(R.id.login));
        //Wait for activity: 'edu.fiu.hmts_cu.activity.MenuActivity'
    assertTrue("edu.fiu.hmts_cu.activity.MenuActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.MenuActivity.class));
        //Click on Empty Text View
    solo.clickOnView(solo.getView(R.id.prodadd, 1));
    assertEquals("1", ((TextView)solo.getView(R.id.prodquatity, 1)).getText());
}


public void testRun() {
        //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
    solo.waitForActivity(SplashActivity.class, 2000);
        //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
    assertTrue("edu.fiu.hmts_cu.activity.LoginActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class));
        //Enter the text: 'jack@fiu.edu'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
    solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "bob@fiu.edu");
        //Enter the text: 'jk1234'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
    solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "bb1234");
        //Click on Log In
    solo.clickOnView(solo.getView(R.id.login));
        //Wait for activity: 'edu.fiu.hmts_cu.activity.MenuActivity'
    assertTrue("edu.fiu.hmts_cu.activity.MenuActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.MenuActivity.class));
        //Click on Empty Text View
    for (int i = 0; i < 25; i++)
      solo.clickOnView(solo.getView(R.id.prodadd));
    assertEquals("20", ((TextView)solo.getView(R.id.prodquatity)).getText());
}
```

- **RemoveProductTest**

```
package edu.fiu.hmts_cu;

import edu.fiu.hmts_cu.activity.SplashActivity;
import com.robotium.solo.*;
import android.test.ActivityInstrumentationTestCase2;
import android.widget.TextView;

/**
 * Test RemoveProduct.
```

```
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
public class testRemoveProduct_S1 extends
ActivityInstrumentationTestCase2<SplashActivity> {
  private Solo solo;

  public testRemoveProduct_S1() {
    super(SplashActivity.class);
  }

  public void setUp() throws Exception {
      super.setUp();
    solo = new Solo(getInstrumentation());
    getActivity();
  }

     @Override
     public void tearDown() throws Exception {
      solo.finishOpenedActivities();
      super.tearDown();
  }

  public void testRun() {
     //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
    solo.waitForActivity(SplashActivity.class, 2000);
     //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
    solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class);
     //Enter the text: 'jack@fiu.edu'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
    solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "jack@fiu.edu");
     //Enter the text: 'jk1234'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
    solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "jk1234");
     //Click on Log In
    solo.clickOnView(solo.getView(R.id.login));
     //Wait for activity: 'edu.fiu.hmts_cu.activity.MenuActivity'
    assertTrue("edu.fiu.hmts_cu.activity.MenuActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.MenuActivity.class));
     //Click on Empty Text View
    solo.clickOnView(solo.getView(R.id.prodadd));
     //Click on Empty Text View
    solo.clickOnView(solo.getView(R.id.prodminus));
    assertEquals("0", ((TextView)solo.getView(R.id.prodquatity)).getText());
  }
}
```

```
public void testRun() {
    //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
  solo.waitForActivity(SplashActivity.class, 2000);
    //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
  solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class);
    //Enter the text: 'jack@fiu.edu'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
  solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "jack@fiu.edu");
    //Enter the text: 'jk1234'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
  solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "jk1234");
    //Click on Log In
  solo.clickOnView(solo.getView(R.id.login));
    //Wait for activity: 'edu.fiu.hmts_cu.activity.MenuActivity'
  assertTrue("edu.fiu.hmts_cu.activity.MenuActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.MenuActivity.class));
    //Click on Empty Text View
  solo.clickOnView(solo.getView(R.id.prodadd, 1));
    //Click on Empty Text View
  solo.clickOnView(solo.getView(R.id.prodminus, 1));
  assertEquals("0", ((TextView)solo.getView(R.id.prodquatity)).getText());
}
```

```
public void testRun() {
    //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
  solo.waitForActivity(SplashActivity.class, 2000);
    //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
  solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class);
    //Enter the text: 'jack@fiu.edu'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
  solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "jack@fiu.edu");
    //Enter the text: 'jk1234'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
  solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "jk1234");
    //Click on Log In
  solo.clickOnView(solo.getView(R.id.login));
    //Wait for activity: 'edu.fiu.hmts_cu.activity.MenuActivity'
  assertTrue("edu.fiu.hmts_cu.activity.MenuActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.MenuActivity.class));
    //Click on Empty Text View
  for (int i = 0; i < 2; i++)
    solo.clickOnView(solo.getView(R.id.prodadd));
    //Click on Empty Text View
  for (int i = 0; i < 3; i++)
    solo.clickOnView(solo.getView(R.id.prodminus));
  assertEquals("0", ((TextView)solo.getView(R.id.prodquatity)).getText());
}
```

- **ViewCartTest**

```
package edu.fiu.hmts_cu;

import android.test.ActivityInstrumentationTestCase2;
import android.widget.TextView;

import com.robotium.solo.Solo;

import edu.fiu.hmts_cu.activity.SplashActivity;

/**
 * Test RemoveProduct.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
public class testViewCart_S1 extends ActivityInstrumentationTestCase2<SplashActivity> {
  private Solo solo;

  public testViewCart_S1() {
    super(SplashActivity.class);
  }

  public void setUp() throws Exception {
      super.setUp();
    solo = new Solo(getInstrumentation());
    getActivity();
  }

     @Override
     public void tearDown() throws Exception {
      solo.finishOpenedActivities();
      super.tearDown();
  }

  public void testRun() {
      //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
    solo.waitForActivity(SplashActivity.class, 2000);
      //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
    solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class);
      //Enter the text: 'jack@fiu.edu'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
    solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "jack@fiu.edu");
      //Enter the text: 'jk1234'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
    solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "jk1234");
      //Click on Log In
```

```
    solo.clickOnView(solo.getView(R.id.login));
     //Wait for activity: 'edu.fiu.hmts_cu.activity.MenuActivity'
    solo.waitForActivity(edu.fiu.hmts_cu.activity.MenuActivity.class);
     //Click on Empty Text View
    solo.clickOnView(solo.getView(R.id.prodadd));
    //Click on Empty Text View
    solo.clickOnView(solo.getView(edu.fiu.hmts_cu.R.id.header_right_btn));
    //Wait for activity: 'edu.fiu.hmts_cu.activity.CartActivity'
    assertTrue("edu.fiu.hmts_cu.activity.CartActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.CartActivity.class));
    assertEquals("Turkey Breast", ((TextView)solo.getView(R.id.cartname,1)).getText());
    assertEquals("Dish", ((TextView)solo.getView(R.id.carttype,1)).getText());
    assertEquals("8.99", ((TextView)solo.getView(R.id.cartprice,1)).getText());
    assertEquals("1", ((TextView)solo.getView(R.id.cartquatity,1)).getText());
    assertEquals("8.99", ((TextView)solo.getView(R.id.carttotal,1)).getText());
    assertEquals("TOTAL: $8.99 USD", ((TextView)solo.getView(R.id.total)).getText());
  }
}

public void testRun() {
    //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
  solo.waitForActivity(SplashActivity.class, 2000);
    //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
  solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class);
    //Enter the text: 'jack@fiu.edu'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
  solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "jack@fiu.edu");
    //Enter the text: 'jk1234'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
  solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "jk1234");
    //Click on Log In
  solo.clickOnView(solo.getView(R.id.login));
    //Wait for activity: 'edu.fiu.hmts_cu.activity.MenuActivity'
  solo.waitForActivity(edu.fiu.hmts_cu.activity.MenuActivity.class);
    //Click on Empty Text View
  solo.clickOnView(solo.getView(R.id.prodadd));
  solo.clickOnView(solo.getView(R.id.prodadd, 1));
  //Click on Empty Text View
  solo.clickOnView(solo.getView(R.id.header_right_btn));
  //Wait for activity: 'edu.fiu.hmts_cu.activity.CartActivity'
  assertTrue("edu.fiu.hmts_cu.activity.CartActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.CartActivity.class));
  assertEquals("Turkey Breast", ((TextView)solo.getView(R.id.cartname,1)).getText());
  assertEquals("Lemonade", ((TextView)solo.getView(R.id.cartname,2)).getText());
  assertEquals("Dish", ((TextView)solo.getView(R.id.carttype,1)).getText());
  assertEquals("Drink", ((TextView)solo.getView(R.id.carttype,2)).getText());
  assertEquals("8.99", ((TextView)solo.getView(R.id.cartprice,1)).getText());
  assertEquals("1.99", ((TextView)solo.getView(R.id.cartprice,2)).getText());
```

```
  assertEquals("1", ((TextView)solo.getView(R.id.cartquatity,1)).getText());
  assertEquals("1", ((TextView)solo.getView(R.id.cartquatity,2)).getText());
  assertEquals("8.99", ((TextView)solo.getView(R.id.carttotal,1)).getText());
  assertEquals("1.99", ((TextView)solo.getView(R.id.carttotal,2)).getText());
  assertEquals("TOTAL: $10.98 USD", ((TextView)solo.getView(R.id.total)).getText());
}

public void testRun() {
    //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
  solo.waitForActivity(SplashActivity.class, 2000);
    //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
  solo.waitForActivity(LoginActivity.class);
    //Enter the text: 'jack@fiu.edu'
  solo.clearEditText((EditText) solo.getView(R.id.usernm));
  solo.enterText((EditText) solo.getView(R.id.usernm), "jack@fiu.edu");
    //Enter the text: 'jk1234'
  solo.clearEditText((EditText) solo.getView(R.id.passwd));
  solo.enterText((EditText) solo.getView(R.id.passwd), "jk1234");
    //Click on Log In
  solo.clickOnView(solo.getView(R.id.login));
    //Wait for activity: 'edu.fiu.hmts_cu.activity.MenuActivity'
  solo.waitForActivity(MenuActivity.class);
    //Click on Empty Text View
  solo.clickOnView(solo.getView(R.id.header_right_btn));
  assertTrue("Could not find the dialog!", solo.searchText("Sorry! Shopping cart is
empty"));
}
```

- **FillOutOrderInfoTest**

```
package edu.fiu.hmts_cu;

import android.test.ActivityInstrumentationTestCase2;

import com.robotium.solo.Solo;

import edu.fiu.hmts_cu.activity.SplashActivity;

/**
 * Test FilloutOrderInfo(Card).
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
public class testFillOutOrderInfo_Card_S1 extends
ActivityInstrumentationTestCase2<SplashActivity> {
  private Solo solo;
```

```
  public testFillOutOrderInfo_Card_S1() {
    super(SplashActivity.class);
  }

  public void setUp() throws Exception {
      super.setUp();
    solo = new Solo(getInstrumentation());
    getActivity();
  }

     @Override
     public void tearDown() throws Exception {
      solo.finishOpenedActivities();
      super.tearDown();
  }

  public void testRun() {
      //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
    solo.waitForActivity(SplashActivity.class, 2000);
      //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
    assertTrue("edu.fiu.hmts_cu.activity.LoginActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class));
      //Enter the text: 'jack@fiu.edu'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
    solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "bob@fiu.edu");
      //Enter the text: 'jk1234'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
    solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "bb1234");
      //Click on Log In
    solo.clickOnView(solo.getView(R.id.login));
      //Wait for activity: 'edu.fiu.hmts_cu.activity.MenuActivity'
    solo.waitForActivity(edu.fiu.hmts_cu.activity.MenuActivity.class);

    //Click on Empty Text View
    solo.clickOnView(solo.getView(R.id.prodadd));
    //Click on Empty Text View
    solo.clickOnView(solo.getView(R.id.header_right_btn));
    //Wait for activity: 'edu.fiu.hmts_cu.activity.CartActivity'
    solo.waitForActivity(edu.fiu.hmts_cu.activity.CartActivity.class);
    //Click on Empty Text View
    solo.clickOnView(solo.getView(R.id.header_right_btn));

    //Wait for activity: 'edu.fiu.hmts_cu.activity.OrderInfoActivity'
    assertTrue("edu.fiu.hmts_cu.activity.OrderInfoActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.OrderInfoActivity.class));
    //Enter the text: '120 SW 7th St'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.shipaddr));
```

```
    solo.enterText((android.widget.EditText) solo.getView(R.id.shipaddr), "120 SW 7th St");
    //Enter the text: 'Miami, FL 33155'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.shipcity));
    solo.enterText((android.widget.EditText) solo.getView(R.id.shipcity), "Miami, FL 33155");
    //Enter the text: 'Please fast'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.deliverynote));
    solo.enterText((android.widget.EditText) solo.getView(R.id.deliverynote), "Please
quickly.");
    //Click on Pay by Debit/Credit Card
    solo.clickOnView(solo.getView(edu.fiu.hmts_cu.R.id.card));
    //Click on Empty Text View
    solo.clickOnView(solo.getView(edu.fiu.hmts_cu.R.id.header_right_btn));
    //Wait for activity: 'edu.fiu.hmts_cu.activity.CardActivity'
    assertTrue("edu.fiu.hmts_cu.activity.CardActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.CardActivity.class));

    //Enter the text: '8756546526590677'
    solo.clearEditText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.cardnum));
    solo.enterText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.cardnum),
"8756546526590677");
    //Enter the text: 'Bob'
    solo.clearEditText((android.widget.EditText)
solo.getView(edu.fiu.hmts_cu.R.id.nameoncard));
    solo.enterText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.nameoncard),
"Bob");
    //Enter the text: '05/2020'
    solo.clearEditText((android.widget.EditText)
solo.getView(edu.fiu.hmts_cu.R.id.expiration));
    solo.enterText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.expiration),
"05/2020");
    //Enter the text: '343'
    solo.clearEditText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.seccode));
    solo.enterText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.seccode),
"343");
    //Click on Empty Text View
    solo.clickOnView(solo.getView(edu.fiu.hmts_cu.R.id.header_right_btn));
    //Wait for activity: 'edu.fiu.hmts_cu.activity.CartActivity'
    assertTrue("edu.fiu.hmts_cu.activity.CartActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.CartActivity.class));
    //Click on Empty Text View
    solo.clickOnView(solo.getView(edu.fiu.hmts_cu.R.id.header_left_btn));
    //Click on Place Order
    solo.clickOnView(solo.getView(edu.fiu.hmts_cu.R.id.placeordercard));
    //Wait for activity: 'edu.fiu.hmts_cu.activity.PaymentActivity'
    assertTrue("edu.fiu.hmts_cu.activity.PaymentActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.PaymentActivity.class));
    assertTrue("Confirmation is not found!", solo.searchText("We will process your order
recently."));
```

```
  }
}

public void testRun() {
    //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
  solo.waitForActivity(SplashActivity.class, 2000);
    //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
  assertTrue("edu.fiu.hmts_cu.activity.LoginActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class));
    //Enter the text: 'jack@fiu.edu'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
  solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "jack@fiu.edu");
    //Enter the text: 'jk1234'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
  solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "jk1234");
    //Click on Log In
  solo.clickOnView(solo.getView(R.id.login));
    //Wait for activity: 'edu.fiu.hmts_cu.activity.MenuActivity'
  solo.waitForActivity(edu.fiu.hmts_cu.activity.MenuActivity.class);

  //Click on Empty Text View
  solo.clickOnView(solo.getView(R.id.prodadd));
  solo.clickOnView(solo.getView(R.id.prodadd, 1));
  solo.clickOnView(solo.getView(R.id.prodadd, 2));
  //Click on Empty Text View
  solo.clickOnView(solo.getView(R.id.header_right_btn));
  //Wait for activity: 'edu.fiu.hmts_cu.activity.CartActivity'
  solo.waitForActivity(edu.fiu.hmts_cu.activity.CartActivity.class);
  //Click on Empty Text View
  solo.clickOnView(solo.getView(R.id.header_right_btn));

  //Wait for activity: 'edu.fiu.hmts_cu.activity.OrderInfoActivity'
  assertTrue("edu.fiu.hmts_cu.activity.OrderInfoActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.OrderInfoActivity.class));
  //Enter the text: '120 SW 7th St'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.shipaddr));
  solo.enterText((android.widget.EditText) solo.getView(R.id.shipaddr), "110 NW 22th Ave");
  //Enter the text: 'Miami, FL 33155'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.shipcity));
  solo.enterText((android.widget.EditText) solo.getView(R.id.shipcity), "Miami, FL 33109");
  //Enter the text: 'Please fast'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.deliverynote));
  solo.enterText((android.widget.EditText) solo.getView(R.id.deliverynote), "Thank you.");
  //Click on Pay by Debit/Credit Card
  solo.clickOnView(solo.getView(R.id.card));
  //Click on Empty Text View
  solo.clickOnView(solo.getView(R.id.header_right_btn));
  //Wait for activity: 'edu.fiu.hmts_cu.activity.CardActivity'
```

```
   assertTrue("edu.fiu.hmts_cu.activity.CardActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.CardActivity.class));

   //Enter the text: '8756546526590677'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.cardnum));
   solo.enterText((android.widget.EditText) solo.getView(R.id.cardnum),
"8457874367235542");
   //Enter the text: 'Bob'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.nameoncard));
   solo.enterText((android.widget.EditText) solo.getView(R.id.nameoncard), "Jack");
   //Enter the text: '05/2020'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.expiration));
   solo.enterText((android.widget.EditText) solo.getView(R.id.expiration), "09/2025");
   //Enter the text: '343'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.seccode));
   solo.enterText((android.widget.EditText) solo.getView(R.id.seccode), "456");
   //Click on Empty Text View
   solo.clickOnView(solo.getView(R.id.header_right_btn));
   //Wait for activity: 'edu.fiu.hmts_cu.activity.CartActivity'
   assertTrue("edu.fiu.hmts_cu.activity.CartActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.CartActivity.class));
   //Click on Empty Text View
   solo.clickOnView(solo.getView(R.id.header_left_btn));
   //Click on Place Order
   solo.clickOnView(solo.getView(R.id.placeordercard));
   //Wait for activity: 'edu.fiu.hmts_cu.activity.PaymentActivity'
   assertTrue("edu.fiu.hmts_cu.activity.PaymentActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.PaymentActivity.class));
   assertTrue("Confirmation is not found!", solo.searchText("We will process your order
recently."));
}


public void testRun() {
      //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
   solo.waitForActivity(SplashActivity.class, 2000);
      //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
   assertTrue("edu.fiu.hmts_cu.activity.LoginActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class));
      //Enter the text: 'jack@fiu.edu'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
   solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "bob@fiu.edu");
      //Enter the text: 'jk1234'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
   solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "bb1234");
      //Click on Log In
   solo.clickOnView(solo.getView(R.id.login));
      //Wait for activity: 'edu.fiu.hmts_cu.activity.MenuActivity'
   solo.waitForActivity(edu.fiu.hmts_cu.activity.MenuActivity.class);
```

```
  //Click on Empty Text View
  solo.clickOnView(solo.getView(R.id.prodadd));
  //Click on Empty Text View
  solo.clickOnView(solo.getView(R.id.header_right_btn));
  //Wait for activity: 'edu.fiu.hmts_cu.activity.CartActivity'
  solo.waitForActivity(edu.fiu.hmts_cu.activity.CartActivity.class);
  //Click on Empty Text View
  solo.clickOnView(solo.getView(R.id.header_right_btn));

  //Wait for activity: 'edu.fiu.hmts_cu.activity.OrderInfoActivity'
  assertTrue("edu.fiu.hmts_cu.activity.OrderInfoActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.OrderInfoActivity.class));
  //Enter the text: '120 SW 7th St'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.shipaddr));
  solo.enterText((android.widget.EditText) solo.getView(R.id.shipaddr), "120 SW 7th St");
  //Enter the text: 'Miami, FL 33155'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.shipcity));
  solo.enterText((android.widget.EditText) solo.getView(R.id.shipcity), "Miami, FL 33155");
  //Enter the text: 'Please fast'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.deliverynote));
  solo.enterText((android.widget.EditText) solo.getView(R.id.deliverynote), "Please quickly.");
  //Click on Pay by Debit/Credit Card
  solo.clickOnView(solo.getView(R.id.card));
  //Click on Empty Text View
  solo.clickOnView(solo.getView(R.id.header_right_btn));
  //Wait for activity: 'edu.fiu.hmts_cu.activity.CardActivity'
  assertTrue("edu.fiu.hmts_cu.activity.CardActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.CardActivity.class));

  //Enter the text: 'Bob'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.nameoncard));
  solo.enterText((android.widget.EditText) solo.getView(R.id.nameoncard), "Bob");
  //Enter the text: '05/2020'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.expiration));
  solo.enterText((android.widget.EditText) solo.getView(R.id.expiration), "05/2020");
  //Enter the text: '343'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.seccode));
  solo.enterText((android.widget.EditText) solo.getView(R.id.seccode), "343");
  //Click on Place Order
  solo.clickOnView(solo.getView(R.id.placeordercard));
  //Wait for activity: 'edu.fiu.hmts_cu.activity.PaymentActivity'
  assertTrue("Text is not found!", solo.searchText("Sorry! The card information is
incomplete."));
}
```

- **PopAddressAlertTest**

```
package edu.fiu.hmts_cu;

import android.test.ActivityInstrumentationTestCase2;

import com.robotium.solo.Solo;

import edu.fiu.hmts_cu.activity.SplashActivity;

/**
 * Test PopAddressAlert.
 *
 * @author  Wenbo Wang
 * @version 1.0, April 2016
 */
public class testPopAddressAlert_S1 extends
ActivityInstrumentationTestCase2<SplashActivity> {
  private Solo solo;

  public testPopAddressAlert_S1() {
    super(SplashActivity.class);
  }

  public void setUp() throws Exception {
      super.setUp();
    solo = new Solo(getInstrumentation());
    getActivity();
  }

     @Override
     public void tearDown() throws Exception {
      solo.finishOpenedActivities();
      super.tearDown();
  }

  public void testRun() {
      //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
    solo.waitForActivity(SplashActivity.class, 2000);
      //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
    assertTrue("edu.fiu.hmts_cu.activity.LoginActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class));
      //Enter the text: 'jack@fiu.edu'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
    solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "bob@fiu.edu");
      //Enter the text: 'jk1234'
    solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
    solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "bb1234");
```

```
        //Click on Log In
        solo.clickOnView(solo.getView(R.id.login));
        //Wait for activity: 'edu.fiu.hmts_cu.activity.MenuActivity'
        solo.waitForActivity(edu.fiu.hmts_cu.activity.MenuActivity.class);

        //Click on Empty Text View
        solo.clickOnView(solo.getView(R.id.prodadd));
        //Click on Empty Text View
        solo.clickOnView(solo.getView(R.id.header_right_btn));
        //Wait for activity: 'edu.fiu.hmts_cu.activity.CartActivity'
        solo.waitForActivity(edu.fiu.hmts_cu.activity.CartActivity.class);
        //Click on Empty Text View
        solo.clickOnView(solo.getView(R.id.header_right_btn));

        //Wait for activity: 'edu.fiu.hmts_cu.activity.OrderInfoActivity'
        assertTrue("edu.fiu.hmts_cu.activity.OrderInfoActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.OrderInfoActivity.class));
        //Enter the text: '120 SW 7th St'
        solo.clearEditText((android.widget.EditText) solo.getView(R.id.shipaddr));
        solo.enterText((android.widget.EditText) solo.getView(R.id.shipaddr), "120 SW 7th St");
        //Enter the text: 'Miami, FL 33155'
        solo.clearEditText((android.widget.EditText) solo.getView(R.id.shipcity));
        solo.enterText((android.widget.EditText) solo.getView(R.id.shipcity), "Miami, FL 33155");
        //Enter the text: 'Please fast'
        solo.clearEditText((android.widget.EditText) solo.getView(R.id.deliverynote));
        solo.enterText((android.widget.EditText) solo.getView(R.id.deliverynote), "Please
quickly.");
        //Click on Pay by Debit/Credit Card
        solo.clickOnView(solo.getView(R.id.card));
        //Click on Empty Text View
        solo.clickOnView(solo.getView(R.id.header_right_btn));
        //Wait for activity: 'edu.fiu.hmts_cu.activity.CardActivity'
        assertTrue("edu.fiu.hmts_cu.activity.CardActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.CardActivity.class));

        //Enter the text: '8756546526590677'
        solo.clearEditText((android.widget.EditText) solo.getView(R.id.cardnum));
        solo.enterText((android.widget.EditText) solo.getView(R.id.cardnum),
"8756546526590677");
        //Enter the text: 'Bob'
        solo.clearEditText((android.widget.EditText) solo.getView(R.id.nameoncard));
        solo.enterText((android.widget.EditText) solo.getView(R.id.nameoncard), "Bob");
        //Enter the text: '05/2020'
        solo.clearEditText((android.widget.EditText) solo.getView(R.id.expiration));
        solo.enterText((android.widget.EditText) solo.getView(R.id.expiration), "05/2020");
        //Enter the text: '343'
        solo.clearEditText((android.widget.EditText) solo.getView(R.id.seccode));
        solo.enterText((android.widget.EditText) solo.getView(R.id.seccode), "343");
```

```
    //Enter the text: '1120 NW 7th St'
    solo.clearEditText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.billaddr));
    solo.enterText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.billaddr), "500
NW 18th Ave");
    //Enter the text: 'mMiami, FL 33155'
    solo.clearEditText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.billcity));
    solo.enterText((android.widget.EditText) solo.getView(edu.fiu.hmts_cu.R.id.billcity),
"Miami, FL 33200");
    //Click on Place Order
    solo.clickOnView(solo.getView(edu.fiu.hmts_cu.R.id.placeordercard));
    assertTrue("Text is not found!", solo.searchText("Your shipping address and billing
address are different."));
    //Click on YES
    solo.clickOnView(solo.getView(android.R.id.button1));
    //Wait for activity: 'edu.fiu.hmts_cu.activity.PaymentActivity'
    assertTrue("edu.fiu.hmts_cu.activity.PaymentActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.PaymentActivity.class));
    assertTrue("Confirmation is not found!", solo.searchText("We will process your order
recently."));
  }
}

public void testRun() {
    //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
  solo.waitForActivity(SplashActivity.class, 2000);
    //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
  assertTrue("edu.fiu.hmts_cu.activity.LoginActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class));
    //Enter the text: 'jack@fiu.edu'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
  solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "bob@fiu.edu");
    //Enter the text: 'jk1234'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
  solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "bb1234");
    //Click on Log In
  solo.clickOnView(solo.getView(R.id.login));
    //Wait for activity: 'edu.fiu.hmts_cu.activity.MenuActivity'
  solo.waitForActivity(edu.fiu.hmts_cu.activity.MenuActivity.class);

  //Click on Empty Text View
  solo.clickOnView(solo.getView(R.id.prodadd));
  //Click on Empty Text View
  solo.clickOnView(solo.getView(R.id.header_right_btn));
  //Wait for activity: 'edu.fiu.hmts_cu.activity.CartActivity'
  solo.waitForActivity(edu.fiu.hmts_cu.activity.CartActivity.class);
  //Click on Empty Text View
  solo.clickOnView(solo.getView(R.id.header_right_btn));
```

```
   //Wait for activity: 'edu.fiu.hmts_cu.activity.OrderInfoActivity'
   assertTrue("edu.fiu.hmts_cu.activity.OrderInfoActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.OrderInfoActivity.class));
   //Enter the text: '120 SW 7th St'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.shipaddr));
   solo.enterText((android.widget.EditText) solo.getView(R.id.shipaddr), "220 NW 15th St");
   //Enter the text: 'Miami, FL 33155'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.shipcity));
   solo.enterText((android.widget.EditText) solo.getView(R.id.shipcity), "Miami, FL 33366");
   //Click on Pay by Debit/Credit Card
   solo.clickOnView(solo.getView(R.id.card));
   //Click on Empty Text View
   solo.clickOnView(solo.getView(R.id.header_right_btn));
   //Wait for activity: 'edu.fiu.hmts_cu.activity.CardActivity'
   assertTrue("edu.fiu.hmts_cu.activity.CardActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.CardActivity.class));

   //Enter the text: '8756546526590677'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.cardnum));
   solo.enterText((android.widget.EditText) solo.getView(R.id.cardnum),
"8756546526590677");
   //Enter the text: 'Bob'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.nameoncard));
   solo.enterText((android.widget.EditText) solo.getView(R.id.nameoncard), "Bob");
   //Enter the text: '05/2020'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.expiration));
   solo.enterText((android.widget.EditText) solo.getView(R.id.expiration), "12/2023");
   //Enter the text: '343'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.seccode));
   solo.enterText((android.widget.EditText) solo.getView(R.id.seccode), "765");
   //Enter the text: '1120 NW 7th St'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.billaddr));
   solo.enterText((android.widget.EditText) solo.getView(R.id.billaddr), "222 SW 87th St");
   //Enter the text: 'mMiami, FL 33155'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.billcity));
   solo.enterText((android.widget.EditText) solo.getView(R.id.billcity), "Miami, FL 33178");
   //Click on Place Order
   solo.clickOnView(solo.getView(R.id.placeordercard));
   assertTrue("Text is not found!", solo.searchText("Your shipping address and billing
address are different."));
   //Click on NO
   solo.clickOnView(solo.getView(android.R.id.button2));
}

public void testRun() {
      //Wait for activity: 'edu.fiu.hmts_cu.activity.SplashActivity'
   solo.waitForActivity(SplashActivity.class, 2000);
      //Wait for activity: 'edu.fiu.hmts_cu.activity.LoginActivity'
```

```
   assertTrue("edu.fiu.hmts_cu.activity.LoginActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.LoginActivity.class));
      //Enter the text: 'jack@fiu.edu'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.usernm));
   solo.enterText((android.widget.EditText) solo.getView(R.id.usernm), "bob@fiu.edu");
      //Enter the text: 'jk1234'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.passwd));
   solo.enterText((android.widget.EditText) solo.getView(R.id.passwd), "bb1234");
      //Click on Log In
   solo.clickOnView(solo.getView(R.id.login));
      //Wait for activity: 'edu.fiu.hmts_cu.activity.MenuActivity'
   solo.waitForActivity(edu.fiu.hmts_cu.activity.MenuActivity.class);

   //Click on Empty Text View
   solo.clickOnView(solo.getView(R.id.prodadd));
   //Click on Empty Text View
   solo.clickOnView(solo.getView(R.id.header_right_btn));
   //Wait for activity: 'edu.fiu.hmts_cu.activity.CartActivity'
   solo.waitForActivity(edu.fiu.hmts_cu.activity.CartActivity.class);
   //Click on Empty Text View
   solo.clickOnView(solo.getView(R.id.header_right_btn));

   //Wait for activity: 'edu.fiu.hmts_cu.activity.OrderInfoActivity'
   assertTrue("edu.fiu.hmts_cu.activity.OrderInfoActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.OrderInfoActivity.class));
   //Enter the text: '120 SW 7th St'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.shipaddr));
   solo.enterText((android.widget.EditText) solo.getView(R.id.shipaddr), "120 SW 7th St");
   //Enter the text: 'Miami, FL 33155'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.shipcity));
   solo.enterText((android.widget.EditText) solo.getView(R.id.shipcity), "Miami, FL 33155");
   //Click on Pay by Debit/Credit Card
   solo.clickOnView(solo.getView(R.id.card));
   //Click on Empty Text View
   solo.clickOnView(solo.getView(R.id.header_right_btn));
   //Wait for activity: 'edu.fiu.hmts_cu.activity.CardActivity'
   assertTrue("edu.fiu.hmts_cu.activity.CardActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.CardActivity.class));

   //Enter the text: '8756546526590677'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.cardnum));
   solo.enterText((android.widget.EditText) solo.getView(R.id.cardnum),
"8756546526590677");
   //Enter the text: 'Bob'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.nameoncard));
   solo.enterText((android.widget.EditText) solo.getView(R.id.nameoncard), "Bob");
   //Enter the text: '05/2020'
   solo.clearEditText((android.widget.EditText) solo.getView(R.id.expiration));
```

```
  solo.enterText((android.widget.EditText) solo.getView(R.id.expiration), "05/2020");
  //Enter the text: '343'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.seccode));
  solo.enterText((android.widget.EditText) solo.getView(R.id.seccode), "343");
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.billaddr));
  solo.enterText((android.widget.EditText) solo.getView(R.id.billaddr), "120 SW 7th St");
  //Enter the text: 'mMiami, FL 33155'
  solo.clearEditText((android.widget.EditText) solo.getView(R.id.billcity));
  solo.enterText((android.widget.EditText) solo.getView(R.id.billcity), "Miami, FL 33155");
  solo.clickOnView(solo.getView(R.id.placeordercard));
  //Wait for activity: 'edu.fiu.hmts_cu.activity.PaymentActivity'
  assertTrue("edu.fiu.hmts_cu.activity.PaymentActivity is not found!",
solo.waitForActivity(edu.fiu.hmts_cu.activity.PaymentActivity.class));
  assertTrue("Confirmation is not found!", solo.searchText("We will process your order
recently."));
}
```

11.7. Appendix G – Diary of Meeting and Tasks

In this section we present a collection of notes from our diary of meetings and tasks.

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang *(Leader)*, Jineth Quintero *(Timekeeper)*, Raul Camarca *(Minute Taker)*
Late Arrivals	None
Date & Time	January 12, 2016 19:00 - 19:40
Location	AHC5 Room 212A
Agenda	1. To introduce each other; 2. Assign roles for the first time; 3. Decide course of action pending meeting with Prof, Clarke; 4. Coordinate next meeting time.
Summary	1. We introduced each other. Two students have undergraduate CS background and are pursuing graduate studies in IT. One student has undergraduate computer engineering background and is pursuing graduate studies in CS. One student has previous experience with object modeling and some of the software to be used. We exchanged contact information. 2. Upon a brief conversation we agreed that for this first part of the term Wenbo will be the team leader, Jineth the timekeeper, and Raul the minute taker. 3. Having been notified that we are to meet with Prof. Clarke by Tuesday, 1/19 before class, we agreed that it would be a good course of action for this week to go over the material on Moodle and to look around for online resources in software engineering. 4. Regarding our next team meeting: Raul is not at FIU on Friday 1/15 and the university will be closed on Monday 1/18 due to holiday, therefore we agreed that the best course of action is to meet on Thursday 1/14 before class. We agreed to meet at 18:00. We did not coordinate a place to meet, assuming the meeting place to be the classroom or outside waiting area.
Assigned Tasks	1. No tasks were assigned at this time; 2. All team members to follow the short term course of action as described in the Summary.

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang *(Leader)*, Jineth Quintero *(Timekeeper)*, Raul Camarca *(Minute Taker*), Alex Marrero Pazos, Weitong Liu
Late Arrivals	None
Date & Time	January 14, 2016 19:00 - 19:40
Location	AHC5 Room 212A
Agenda	1. To introduce new team members. 2. To discuss the meeting with Prof. Clarke. 3. To discuss the project description. 4. To develop 4 use cases and 1 security case per team member.
Summary	1. The new team members are Alex Marrero and Weitong Liu. 2. We refined our idea of a restaurant takeout software system. We went over the three phases that will follow, namely requirements, design, and implementation. 3. We focused on producing the requirements: 1) The user shall visualize menu 2) The user shall visualize dishes 3) The user shall visualize description of dishes 4) The user shall modify dishes adding and removing sides 5) The user shall select dishes for purchase 6) The user shall place dishes on hold in a shopping cart 7) The user shall remove dishes from shopping cart 8) The user shall create an account 9) The user shall modify an account 10) The user shall delete an account 11) The user shall add a delivery user account 12) The user shall remove a delivery user account 13) The user shall visualize addresses to deliver 14) The user shall issue a refund 15) The user shall add new dishes to menu 16) The users shall modify dishes in menu 17) The user shall remove dishes from menu 18) The user shall hide dishes in menu 19) The user shall make visible hidden dishes in menu

	20) The user shall specify sides that can be added 21) Login Timeout 22) Login Failure Times 23) Forgotten Username 24) Security Questions 25) Differing Shipping and Billing Address 4. We agreed that the team leader would make an appointment with Prof. Clarke on Tuesday 1/19 at 4:30 PM
Assigned Tasks	1. Team leader to schedule appointment with Prof. Clarke on 1/18 at 4:30 PM 2. All team members to meet in Prof. Clarke's office by meeting time.

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang *(Leader)*, Jineth Quintero *(Timekeeper)*, Alex Marrero Pazos
Late Arrivals	None
Date & Time	January 19, 2016 19:10 - 19:40
Location	AHC5 Room 212A
Agenda	1. Confirm the use cases for our project. 2. Assign use cases to each team member.
Summary	1. We have 16 use cases and 4 security use cases to write. Please see file Team2 Project Use Cases Confirmation.docx for the details. 2. Assign use cases to each team member, Please see file Phase1_Task_List.xlsx for the details. 3. Finish at least one use case using the template on moodle and upload it before Thursday's meeting (Jan 21th), and we'll check and discuss the use cases for each member on the meeting.
Assigned Tasks	1. Write use cases. Each member needs to write four use cases plus one misuse case and one security use case. The final deadline is 1/26. 2. Use cases should be written and UML diagrams with documentation completed. A team member from each team will present a use case during class. 3. Each team member must create the UML diagrams related to their use cases. Team leader will integrate them.

	4. For the details, please review Project Stuff section in the note CEN5011-notes3_011916.ppt. 5. Coordinate next meeting time.

Project ID	HMTS
Project Name	Happy Meal Take-Out System
Attendance	Wenbo Wang *(Leader)*, Jineth Quintero *(Timekeeper)*, Raul Camarca *(Minute Taker)*, Alex Marrero, Weitong Liu.
Late Arrivals	None
Date & Time	January 21, 2016 19:00 - 19:40
Location	AHC5 Room 212A
Agenda	1. To review development of use cases 2. To plan follow-up meeting before class presentation
Summary	1. Weitong is back in the team. He will begin working on his use cases. 2. We discussed the work done since 1/19, reviewed a use case both in UML graph and textual representation. 3. We reviewed the project deadlines, agreed to finish and upload our individual use cases on Google Drive possibly by the end of Monday, 1/25. 4. We agreed to meet on Tuesday 1/26 at 5 PM to review the material to be presented in class. 5. We also discussed the first interface to be developed. We agreed to use *bootstrap*, an open-source collection of tools for web development, in order to develop a front end framework.
Assigned Tasks	1. All members: finish assigned use cases and UML diagrams, and upload on Google Drive by Tuesday, January 26. 2. Begin to learn *bootstrap* since we will use it to individually produce code for our use cases (official website http://getbootstrap.com) 3. Meet on Tuesday, January 26 at 5 PM.

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang *(Leader)*, Jineth Quintero *(Timekeeper)*, Raul Camarca *(Minute Taker)*, Weitong Liu, Alex Marrero
Late Arrivals	None
Date & Time	January 26, 2016 16:00 – 17:30; 19:30 – 20:30
Location	AHC5 Room 212A
Agenda	1. To review the work carried out in the last week in preparation for in-class presentation. 2. To prepare for a brief Q&A with Prof. Clarke 3. To complete any last-minute refinements for in-class presentation. 4. To discuss the implementation of corrections based on the feedback from professor and classmates. 5. To agree upon the upcoming meeting.
Summary	1. We reviewed the work done and made sure that each team member is familiarized with the work developed by the other members. 2. We reviewed the Papyrus master file on the team leader's laptop so that any team member, if needed, would be able to work on it. 3. We asked Prof. Clarke some questions on the material to be presented to make sure we did not misunderstand or forget anything. 4. We implemented some last-minute presentation components, such as a description of the first and second level Use Case diagrams. 5. We reviewed all use cases after the presentation, discussing what each needs to change based on the feedback received, and what needs to be completed before the next meeting: a. scenarios b. diagrams 6. We agreed to hold our next meeting on Tuesday 2/2 at 4:30 PM in front of Prof. Clarke's office.
Assigned Tasks	1. All team members are to implement global changes and to implement individual changes. 2. All team members are to work on individual tasks as described in summary.

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang *(Leader)*, Jineth Quintero *(Timekeeper)*, Raul Camarca *(Minute Taker)*, Weitong Liu (*Documentation Engineer*), Alex Marrero (*Requirements Engineer*).
Late Arrivals	None
Date & Time	February 2, 2016 16:00 – 17:00; 19:30 – 20:30
Location	AHC5 Room 212A
Agenda	1. To assign new team roles. 2. To review the work carried out in the last week implementing the feedback received during class presentation. 3. To plan on subsequent stage of implementation.
Summary	1. We assigned new team roles: Weitong Liu is Documentation Engineer, Alex Marrero is Requirements Engineer. 2. We reviewed the feedback received during the presentation. 3. We worked on revising the Use Cases and the Use Case diagrams. 4. We extended the changes, when necessary, to Object, Sequence, and Class diagrams. 5. We uploaded our Papyrus models into the Master File. 6. We assigned scenarios to each team member: 1) HMTS_UM01_CreateAccount Alex 2) HMTS_UM02_Login Jineth 3) HMTS_UM04_Logout Jineth 4) HMTS_UM07_RolesControl Weitong 5) HMTS_OP01_VisualizeProducts Jineth 6) HMTS_OP02_AddProduct Wenbo 7) HMTS_CU01_VisualizeMenu Jineth 8) HMTS_CU02_AddProduct Wenbo 9) HMTS_CU03_ViewCart Weitong 10) HMTS_CU04_RemoveProduct Alex 11) HMTS_CU05_FillOutOrderInfo Raul 12) HMTS_CU07_PopAddressAlert Raul

Assigned Tasks	1. All team members are to finalize reviews of their Use Cases and corresponding diagrams. 2. All team members are to work on scenarios in order to bring them to completion by next meeting. 3. We agreed to meet on Thursday 2/4 and review our work.

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang *(Leader)*, Jineth Quintero *(Timekeeper)*, Raul Camarca *(Minute Taker)*, Weitong Liu (*Documentation Engineer*), Alex Marrero (*Requirements Engineer*).
Late Arrivals	None
Date & Time	February 4, 2016 16:00 – 17:00; 19:30 – 20:30
Location	AHC5 Room 212A
Agenda	1. To review the work done on Use Cases. 2. To review the work done on Scenarios. 3. To plan for the following tasks.
Summary	1. We reviewed our updated Use Cases, and the corresponding diagrams. 2. We uploaded our Papyrus models on the Master File. 3. We reviewed the Scenarios developed so far. 4. We agreed that we need to finalize the diagrams and the Scenarios as soon as possible.
Assigned Tasks	1. All team members are to focus exclusively on finalizing the update of their diagrams and uploading the new Papyrus models. 2. All team members are then to finalize their Scenarios. 3. We agreed to meet again on 2/9 and begin working on User Interfaces.

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang *(Leader)*, Jineth Quintero *(Timekeeper)*, Raul Camarca *(Minute Taker)*, Weitong Liu (*Documentation Engineer*), Alex Marrero (*Requirements Engineer*).
Late Arrivals	None
Date & Time	February 9, 2016 16:00 – 17:00; 19:30 – 20:30
Location	AHC5 Room 212A
Agenda	1. To review the work done on the Use Cases and corresponding diagrams. 2. To review the work done on the Scenarios. 3. To discuss and work on User Interfaces.
Summary	1. We reviewed the work carried out by each team member to update the Use Cases and the corresponding diagrams. 2. We reviewed the work done by each team member on their Scenarios. 3. We moved on to discuss and work on the User Interfaces. 4. We assigned UIs to each team member: 1) RegistrationPage Web Alex 2) RegistrationPage Mobile Alex 3) LoginPage Web Alex 4) LoginPage Mobile Alex 5) HomePage Web Wenbo 6) VisualizeProductsPage Web Jineth 7) AddProductPage Web Weitong 8) VisualizeMenuPage Mobile Jineth 9) VisualizeCartPage Mobile Weitong 10) FillOrderInfoPage1 Mobile Raul 11) FillOrderInfoPage2 Mobile Raul 12) FillOrderInfoPage3 Mobile Raul
Assigned Tasks	1. All team members are to finalize their Use Cases, diagrams, and scenarios. 2. All team members are to re-focus on the development of their UIs. 3. We agreed to meet again on 2/11.

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang *(Leader)*, Jineth Quintero *(Timekeeper)*, Raul Camarca *(Minute Taker)*, Weitong Liu (*Documentation Engineer*), Alex Marrero (*Requirements Engineer*).
Late Arrivals	None
Date & Time	February 11, 2016 16:00 – 17:00; 19:30 – 20:30
Location	AHC5 Room 212A
Agenda	1. To review the progression of the updates on the Use Cases, diagrams, and scenarios. 2. To review the work carried out on the User Interfaces. 3. To go over the project timeline to ensure that we are not falling behind. 4. To go over the format of the First Deliverable to ensure that we are building towards the final product consistently. 5. To assess the work to be carried out in evaluations through the COCOMOII model.
Summary	1. We reviewed the update carried out on the Use Cases, diagrams, and scenarios, ensuring consistency throughout the Master File. 2. We reviewed the first drafts on the User Interfaces and agreed to focus more on this aspect of the development. 3. We went over the schedule and realized that we had fallen a bit behind in the development process but well within the progress milestones we had planned. 4. We went over the structure of the First Deliverable and discussed its assembly based on the work carried out by each team member. 5. We discussed the work to be done using the COCOMOII software. Team leader requested to focus on this task.
Assigned Tasks	1. All team members are to continue their work on their UIs. 2. Team leader will work on COCOMOII estimations. 3. We will meet again on 2/16.

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang *(Leader)*, Jineth Quintero *(Timekeeper)*, Raul Camarca *(Minute Taker)*, Weitong Liu (*Documentation Engineer*), Alex Marrero (*Requirements Engineer*).
Late Arrivals	None
Date & Time	February 16, 2016 16:00 – 17:00; 19:30 – 20:30
Location	AHC5 Room 212A
Agenda	1. To review the work on Use Cases, diagrams, Scenarios. 2. To review the work on the software estimation. 3. To review the development of the UIs. 4. To review the assembly of the First Deliverable.
Summary	1. We reviewed the work carried out by each team member on the User Interfaces. 2. We made significant annotations towards establishing a consistency in style between each web and mobile UI, since they were initially developed by the team members independently. 3. We reviewed the work carried out using COCOMOII. 4. We agreed to start assembling the First Deliverable and Powerpoint Presentation, and to hold a meeting on 2/22 to boost this process and finalize the assembly. 5. Team members are to finalize the work on Use Cases, diagrams, Scenarios.
Assigned Tasks	1. All team members are to revise their UIs according to the changes suggested in the meeting. 2. All team members are to begin working on the First Deliverable and Powerpoint Presentation. 3. We will meet again at the Green Library on Monday 2/22.

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang *(Leader)*, Jineth Quintero *(Timekeeper)*, Raul Camarca *(Minute Taker)*, Weitong Liu (*Documentation Engineer*), Alex Marrero (*Requirements Engineer*).
Late Arrivals	None
Date & Time	February 22, 2016 12:00 – 19:00, remotely 21:30-23:30
Location	GL Study Room
Agenda	1. To review the work done on User Interfaces. 2. To complete the First Deliverable. 3. To rehearse the Presentation.
Summary	1. We reviewed the work done on the User Interfaces. 2. We worked on the Papyrus Master File to ensure that all Diagrams were consistent. 3. We assembled the First Deliverable and refinished any work that was pending. 4. We put together the Powerpoint Presentation and rehearsed it based on assignations: 1) Wenbo: Cover, 1.Purpose and Scope, 2.Project Schedule, 3.Cost Estimation 2) Raul: 4.Use Case Diagram for Implementation 3) Alex: 5.Functional Requirements 4) Jineth: 6.Scenario, 7.User Interface 5) Weitong: 8.Object Diagram, 9.Sequence Diagram
Assigned Tasks	1. Minute Taker is to finalize the last Minutes and incorporate them into Appendix D. 2. Minute Taker and Requirements Engineer are to work together on Chapter 1, introduction and summary, and finalize it. 3. Team members are to review their part of the Presentation. 4. We agreed to continue working remotely and to meet again on 2/23.

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang, Jineth Quintero, Raul Camarca, Weitong Liu, Alex Marrero Pazos
Late Arrivals	None
Date & Time	March 1, 2016 16:00 - 17:00
Location	AHC5 Room 212A;
Agenda	1. To recap the presentation and determine tasks for the following weeks
Summary	1. We determined our schedule and work breakdown
Assigned Tasks	1. Determined that we should meet march 3 to finalize tasks

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang, Jineth Quintero, Raul Camarca, Weitong Liu, Alex Marrero Pazos
Late Arrivals	None
Date & Time	March 3, 2016 16:30 - 17:00
Location	ECS Room 212
Agenda	1. To talk about deliverable 1 feedback and assign tasks
Summary	1. In the absence of feedback we decided to just divide up work and assign responsibilities
Assigned Tasks	1. Deployment Diagram- Raul 2. ER Diagram – Wenbo 3. OCL- alex 4. Class interface – wenbo 5. Security management – alex 6. State machines - weitong

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang, Jineth Quintero, Raul Camarca, Weitong Liu, Alex Marrero Pazos
Late Arrivals	None
Date & Time	March 8, 2016 16:30 - 17:00
Location	ECS Room 212
Agenda	1. Discuss ideas on how to move forward with the project
Summary	1. Decided on high-level design decisions such as how we would explain our choices for databases, client interfaces, etc.
Assigned Tasks	1. Deployment Diagram- Raul 2. ER Diagram – Wenbo 3. OCL- alex 4. Class interface – wenbo 5. Security management – alex 6. State machines - weitong

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang, Jineth Quintero, Raul Camarca, Weitong Liu, Alex Marrero Pazos
Late Arrivals	None
Date & Time	March 10, 2016 16:00 - 17:00
Location	ECS Room 212
Agenda	1. To review progress and evaluate feedback
Summary	1. Went through SRD and presentation feedback and established points to improve 2. Assigned Design Document tasks to the different team members 3. Went through the system architecture and design specifications
Assigned Tasks	1. Proposed Software Architecture - Raul 2. Security Management – Alex 3. Detailed Design – Weitong 4. Appendix A – Jineth 5. Appendix B – Jineth 6. Appendix C - Wenbo 7. Appendix D - Wenbo

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang, Jineth Quintero, Raul Camarca, Weitong Liu, Alex Marrero Pazos
Late Arrivals	None
Date & Time	March 15, 2016 10:00 - 13:00
Location	ECS Lab
Agenda	1. To review progress and group parts completed
Summary	1. Went through document requirements and made sure that every group member understood his/her responsibilities 2. Discussed the pertinent aspects of the project and how they could be improved
Assigned Tasks	1. Proposed Software Architecture - Raul 2. Security Management – Alex 3. Detailed Design – Weitong 4. Appendix A – Jineth 5. Appendix B – Jineth 6. Appendix C - Wenbo 7. Appendix D – Wenbo 8. Deployment Diagram- Raul 9. ER Diagram – Wenbo 10. OCL- alex 11. Class interface – wenbo 12. Security management – alex 13. State machines - weitong

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang, Jineth Quintero, Raul Camarca, Weitong Liu, Alex Marrero Pazos
Late Arrivals	None
Date & Time	March 17, 2016 13:00 - 13:30
Location	Phone conference
Agenda	1. To review progress
Summary	1. Reviewed progress and collaborated on assigning final tasks

Assigned Tasks	1. Proposed Software Architecture - Raul 2. Security Management – Alex 3. Detailed Design – Weitong 4. Appendix A – Jineth 5. Appendix B – Jineth 6. Appendix C - Wenbo 7. Appendix D – Wenbo 8. Deployment Diagram- Raul 9. ER Diagram – Wenbo 10. OCL- alex 11. Class interface – wenbo 12. Security management – alex 13. State machines - weitong

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang, Jineth Quintero, Raul Camarca, Weitong Liu, Alex Marrero Pazos
Late Arrivals	None
Date & Time	March 22, 2016 16:00 – 17:00
Location	ECS 212
Agenda	1. To review progress and group parts completed.
Summary	1. Went through DD to check for consistency and fixed last minute errors.
Assigned Tasks	

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang, Jineth Quintero, Raul Camarca, Weitong Liu, Alex Marrero Pazos
Late Arrivals	None
Date & Time	March 29, 2016 16:00 - 17:00
Location	ECS Room 212
Agenda	1. Discuss deliverable 2 already submitted and study for midterm exam
Summary	1. Reviewed together for final exam

Assigned Tasks	

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang, Jineth Quintero, Raul Camarca, Weitong Liu, Alex Marrero Pazos
Late Arrivals	None
Date & Time	April 5, 2016 16:00 - 17:00
Location	ECS Room 212
Agenda	1. Assign tasks for the upcoming phase
Summary	1. Assigned tasks
Assigned Tasks	1. Assigned Raul as Team Leader 2. Assigned SRD and DD correction to members originally responsible for that part

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang, Jineth Quintero, Raul Camarca, Weitong Liu, Alex Marrero Pazos
Late Arrivals	None
Date & Time	April 7, 2016 16:00 - 17:00
Location	ECS Room 212
Agenda	1. To review progress and talk about presentation to professor
Summary	1. Discussed Assignments and discussed the presentation to the professor regarding deliverable 2.
Assigned Tasks	1. Proposed Software Architecture - Raul 2. Security Management – Alex 3. Detailed Design – Weitong 4. Appendix A – Wenbo 5. Appendix B – Jineth 6. Appendix C - Jineth 7. Appendix D – Wenbo 8. Deployment Diagram- Raul 9. ER Diagram – Wenbo 10. OCL- alex 11. Class interface – wenbo

	12. Introduction – Raul 13. Current System – alex 14. Project plan – wenbo 15. System requirements – jineth 16. Project schedule -wenbo

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang, Jineth Quintero, Raul Camarca, Weitong Liu, Alex Marrero Pazos
Late Arrivals	None
Date & Time	April 12, 2016 16:00 - 17:00
Location	ECS Room 212
Agenda	1. To prepare final touches on presentation to professor on deliverable 2 2. To deliver presentation
Summary	1. Presented to professor 2. Got feedback on modifications to the structure of our system 3. Got feedback on organization for the Papyrus model
Assigned Tasks	1. Proposed Software Architecture - Raul 2. Security Management – Alex 3. Detailed Design – Weitong 4. Appendix A – Wenbo 5. Appendix B – Jineth 6. Appendix C - Jineth 7. Appendix D – Wenbo 8. Deployment Diagram- Raul 9. ER Diagram – Wenbo 10. OCL- alex 11. Class interface – wenbo 12. Introduction – Raul 13. Current System – alex 14. Project plan – wenbo 15. System requirements – jineth 16. Project schedule -wenbo

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang, Jineth Quintero, Raul Camarca, Weitong Liu, Alex Marrero Pazos
Late Arrivals	None
Date & Time	April 14, 2016 16:00 - 17:00
Location	ECS Room 212
Agenda	1. Assigning tasks for the presentation
Summary	1. Assigned tasks to presentation. Tasks were in line with what that member was responsible for in the final document 2. Decided that Raul and Wenbo would assemble final document
Assigned Tasks	1. Proposed Software Architecture - Raul 2. Security Management – Alex 3. Detailed Design – Weitong 4. Appendix A – Wenbo 5. Appendix B – Jineth 6. Appendix C - Jineth 7. Appendix D – Wenbo 8. Deployment Diagram- Raul 9. ER Diagram – Wenbo 10. OCL- alex 11. Class interface – wenbo 12. Introduction – Raul 13. Current System – alex 14. Project plan – wenbo 15. System requirements – jineth 16. Project schedule -wenbo

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang, Jineth Quintero, Raul Camarca, Weitong Liu, Alex Marrero Pazos
Late Arrivals	None
Date & Time	April 19, 2016 16:00 - 17:00

Location	ECS Room 212
Agenda	1. Rehearse presentation
Summary	1. Rehearsed presentation and ironed out any details 2. Presented to the class
Assigned Tasks	1. Proposed Software Architecture - Raul 2. Security Management – Alex 3. Detailed Design – Weitong 4. Appendix A – Wenbo 5. Appendix B – Jineth 6. Appendix C - Jineth 7. Appendix D – Wenbo 8. Deployment Diagram- Raul 9. ER Diagram – Wenbo 10. OCL- alex 11. Class interface – wenbo 12. Introduction – Raul 13. Current System – alex 14. Project plan – wenbo 15. System requirements – jineth 16. Project schedule -wenbo

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang, Jineth Quintero, Raul Camarca, Weitong Liu, Alex Marrero Pazos
Late Arrivals	None
Date & Time	April 21, 2016 16:00 - 17:00
Location	ECS Room 212
Agenda	1. Go through test cases both natural language and source
Summary	1. Went through the test cases and found any discrepancies in our testing spectrum 2. Went through the source testing code and made sure it aligned with the natural language test cases
Assigned Tasks	1. Proposed Software Architecture - Raul 2. Security Management – Alex 3. Detailed Design – Weitong

	4. Appendix A – Wenbo 5. Appendix B – Jineth 6. Appendix C - Jineth 7. Appendix D – Wenbo 8. Deployment Diagram- Raul 9. ER Diagram – Wenbo 10. OCL- alex 11. Class interface – wenbo 12. Introduction – Raul 13. Current System – alex 14. Project plan – wenbo 15. System requirements – jineth 16. Project schedule -wenbo

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang, Jineth Quintero, Raul Camarca, Weitong Liu, Alex Marrero Pazos
Late Arrivals	None
Date & Time	April 26, 2016 16:00 - 17:00
Location	ECS Room 212
Agenda	1. Finish up test cases and make sure automation works
Summary	1. Went through test case setup
Assigned Tasks	1. Proposed Software Architecture - Raul 2. Security Management – Alex 3. Detailed Design – Weitong 4. Appendix A – Wenbo 5. Appendix B – Jineth 6. Appendix C - Jineth 7. Appendix D – Wenbo 8. Deployment Diagram- Raul 9. ER Diagram – Wenbo 10. OCL- alex 11. Class interface – wenbo 12. Introduction – Raul 13. Current System – alex

	14. Project plan – wenbo 15. System requirements – jineth

Project ID	HMTS
Project Name	Happy Meal Take-out System
Attendance	Wenbo Wang, Jineth Quintero, Raul Camarca, Weitong Liu, Alex Marrero Pazos
Late Arrivals	None
Date & Time	April 28, 2016 19:25 - 20:40
Location	ECS Room 212
Agenda	1. Finalize final deliverable and source code
Summary	1. Went through final integrated deliverable and made sure everything was consistent with itself and the source
Assigned Tasks	N/A