After the first implementation of the store management system (with a local database), we want to redesign it using a client/server architecture. We can start by rewriting the user stories:

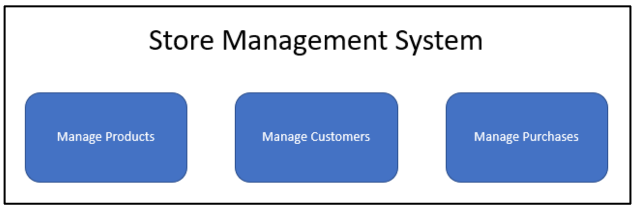
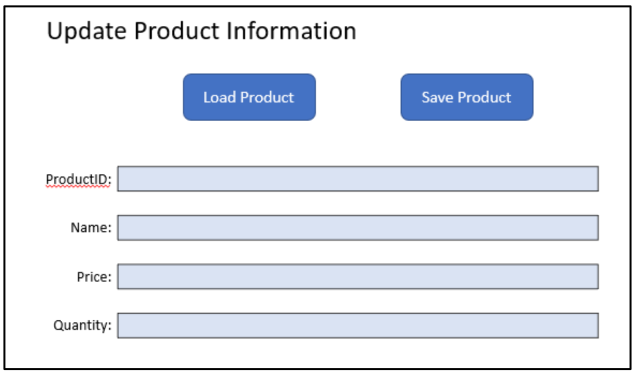
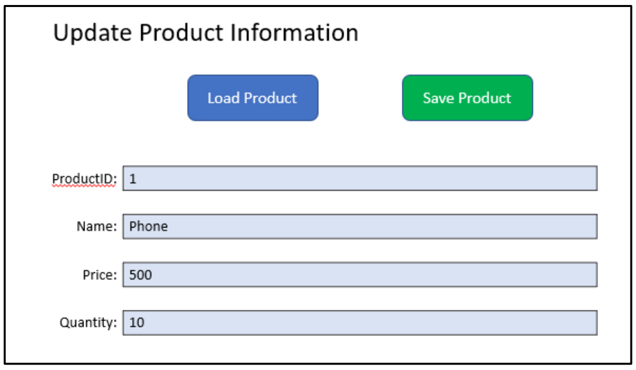
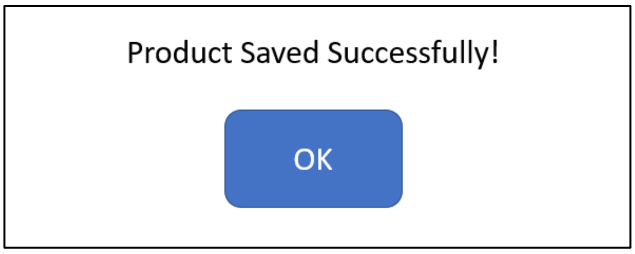
- As a user, I want to add a new product or update a current product in the system.
- As a user, I want to add a new customer or update a current customer in the system.
- As a user, I want to add a new purchase or update a current purchase in the system.

Tasks:

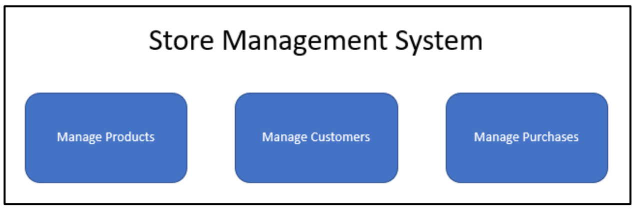
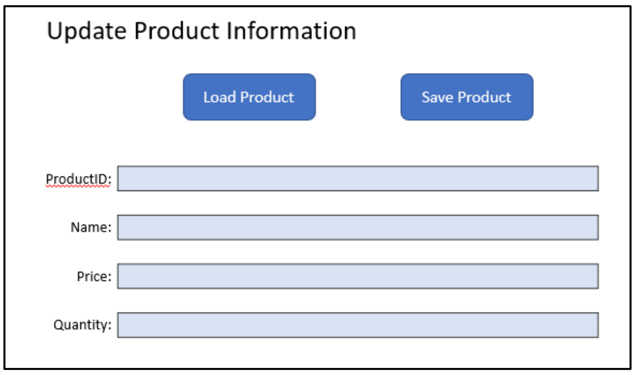
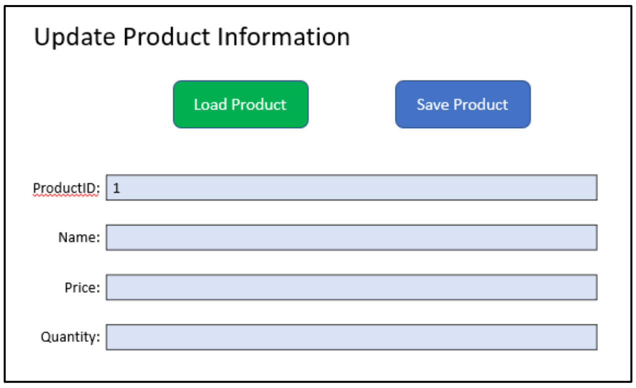
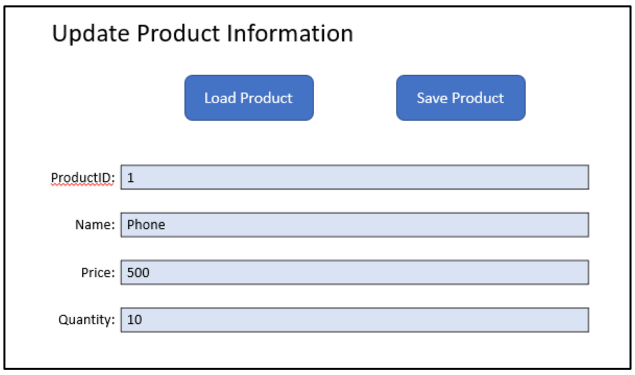
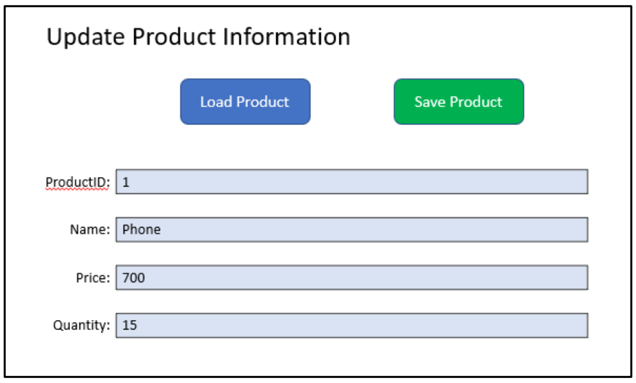
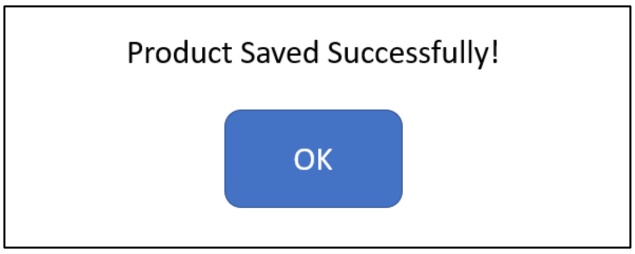
1. Rewrite two common use cases for each user story. Sketch the screens the system should display in each use case.
2. Redesign the Data Access layer at the client side that can load/save data to a remote server component.
3. Design the server component to perform load/save requests from the Data Access layer at the client side. Describe the protocol for two sides: client and server.
4. Implement the system.
5. Test the system with each use case.

Task 1: Rewrite two common use cases for each user story. Sketch the screens the system should display in each use case.

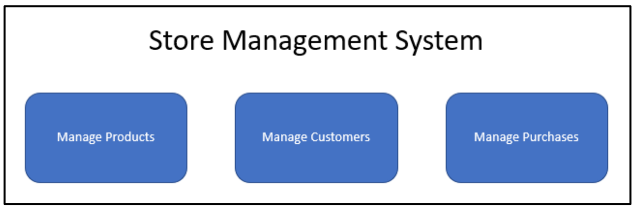
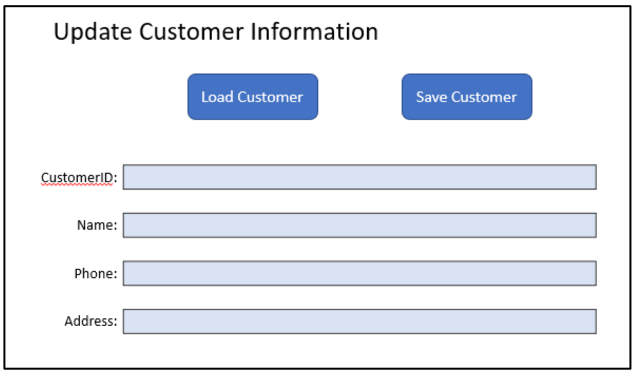
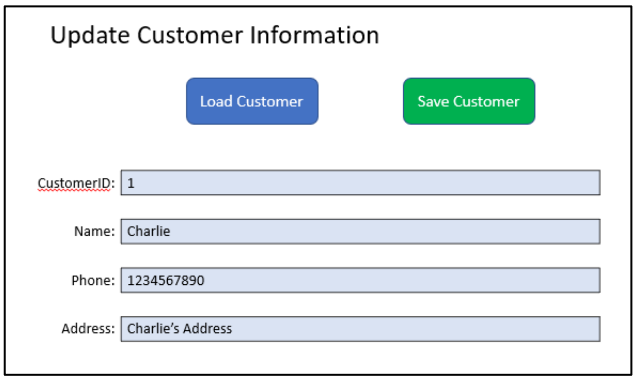
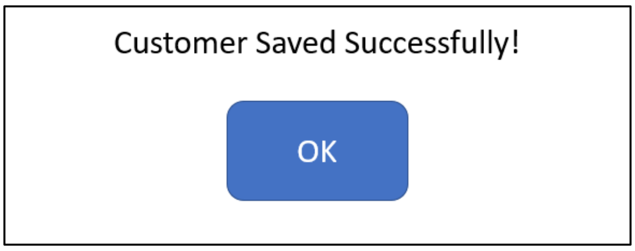
As a user, I want to add a new product to the system

Actor	**System**
1. Choose "Manage Products" on main screen  Store Management System Manage Products Manage Customers Manage Purchases	2. Display "Update Product" screen  Update Product Information Load Product Save Product ProductID: Name: Price: Quantity:
3. Input data then click "Save Product" button  ProductID: 1 Name: Phone Price: 500 Quantity: 10	4. Display "Product Saved Successfully" screen 

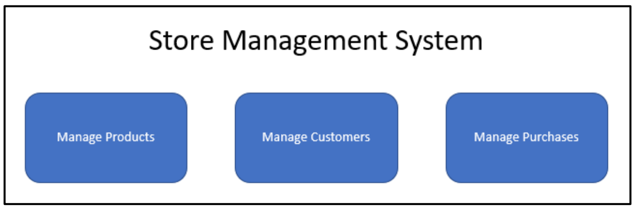
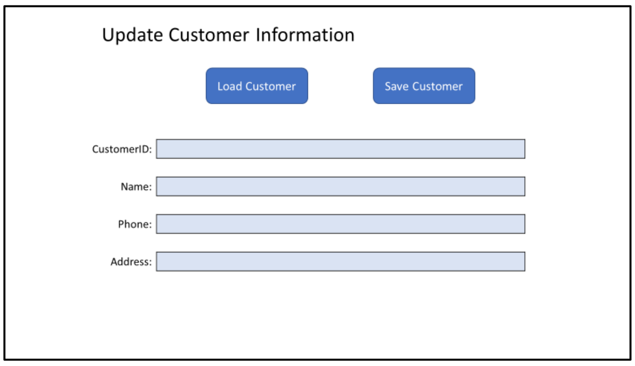
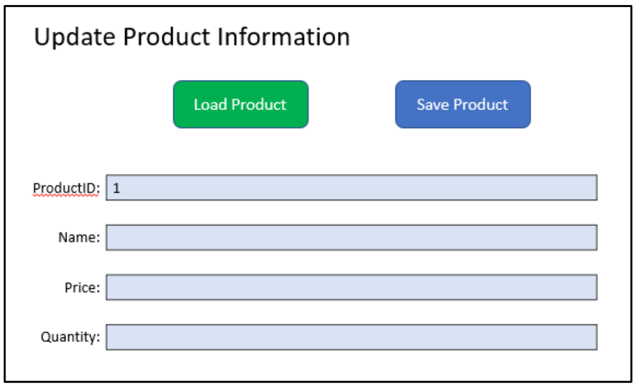
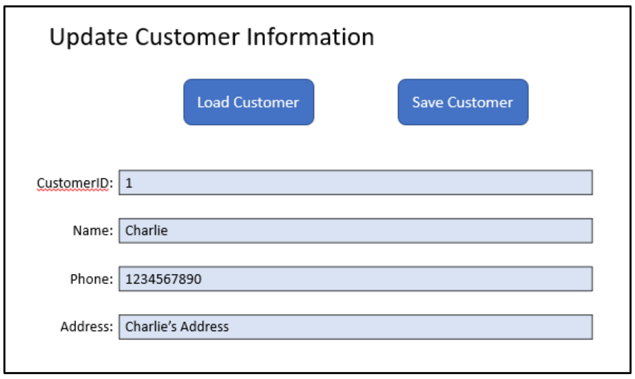
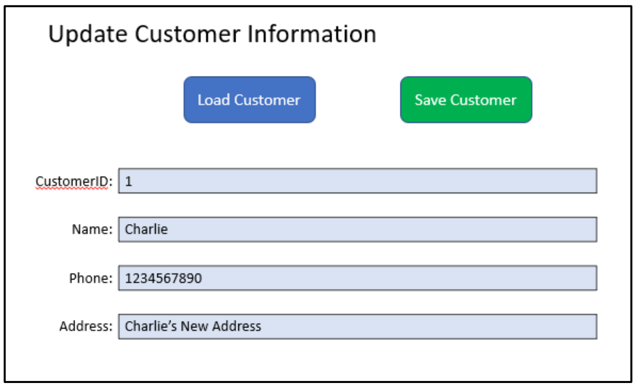
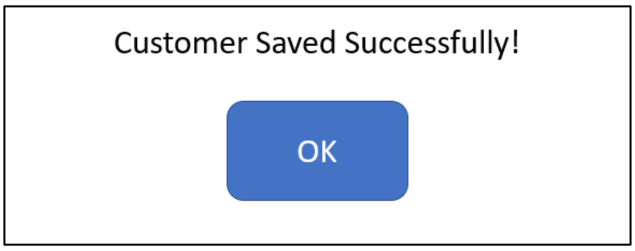
As a user, I want to update a current product in the system.

Actor	**System**
1. Choose "Manage Products" on main screen  Store Management System 	2. Display "Update Product" screen  Update Product Information ProductID: Name: Price: Quantity:
3. Enter ProductID for existing product and click Load Product  Update Product Information ProductID: 1 Name: Price: Quantity:	4. Display Existing Product Information  Update Product Information ProductID: 1 Name: Phone Price: 500 Quantity: 10
5. Update desired info and click Save Product  Update Product Information ProductID: 1 Name: Phone Price: 700 Quantity: 15	6. Display "Product Saved Successfully" screen  Product Saved Successfully!

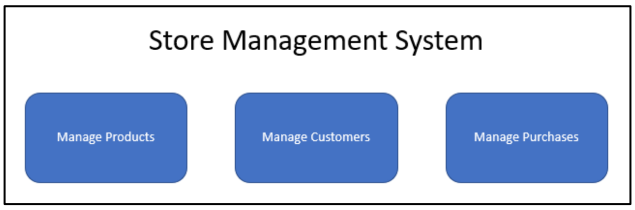
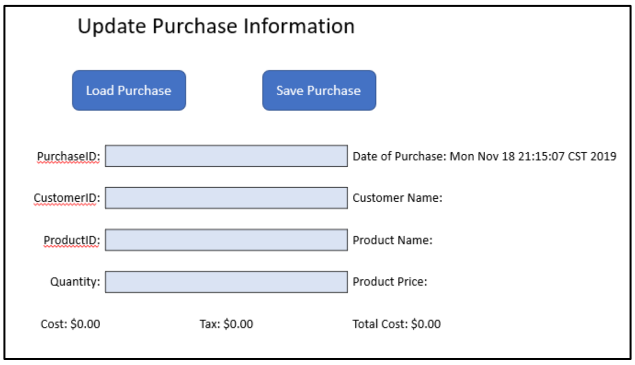
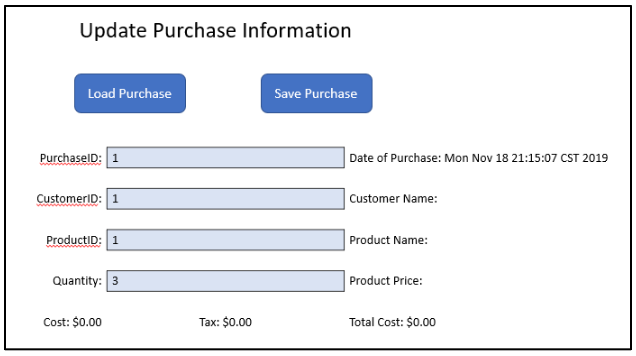
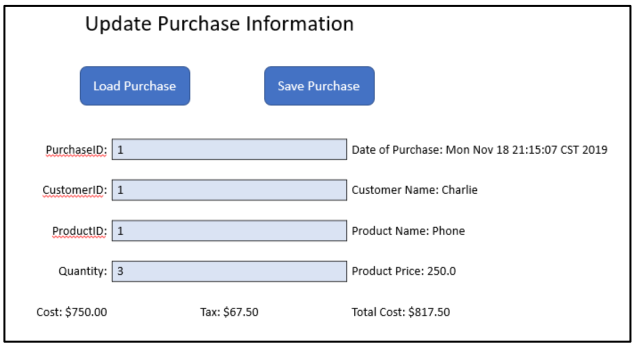
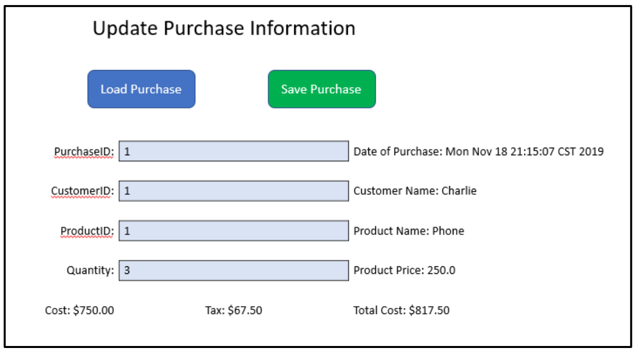
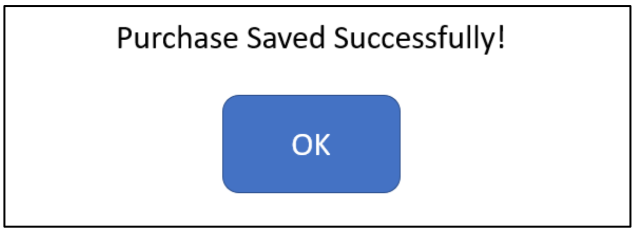
As a user, I want to add a new customer to the system

Actor	**System**
1. Choose "Manage Customers" on main screen  Store Management System Manage Products Manage Customers Manage Purchases	2. Display "Update Customer" screen  Update Customer Information Load Customer Save Customer CustomerID: Name: Phone: Address:
3. Input data then click "Save Customer" button  Update Customer Information Load Customer Save Customer CustomerID: 1 Name: Charlie Phone: 1234567890 Address: Charlie's Address	4. Display "Customer Saved Successfully" screen  Customer Saved Successfully! OK

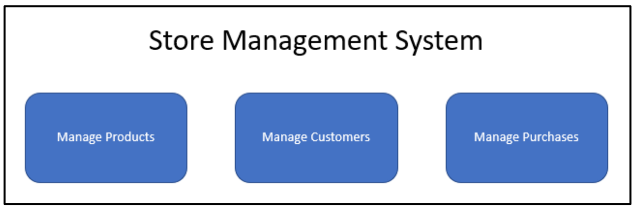
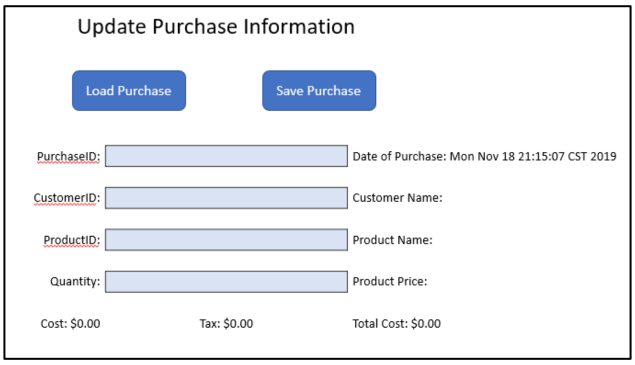
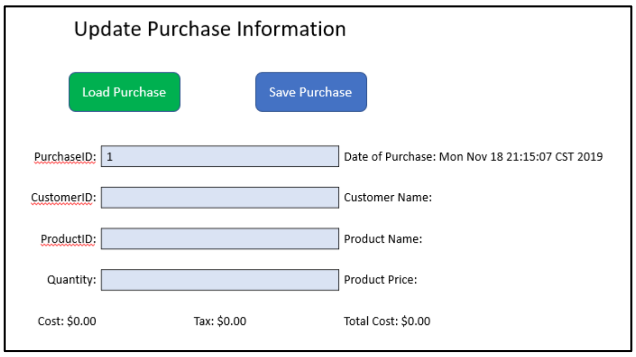
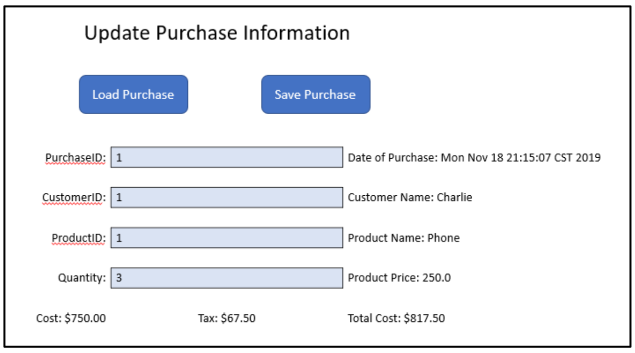
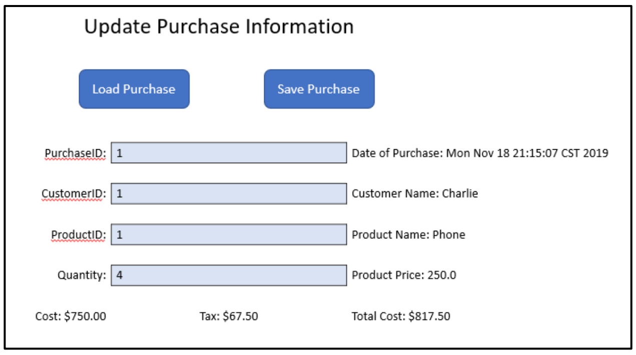
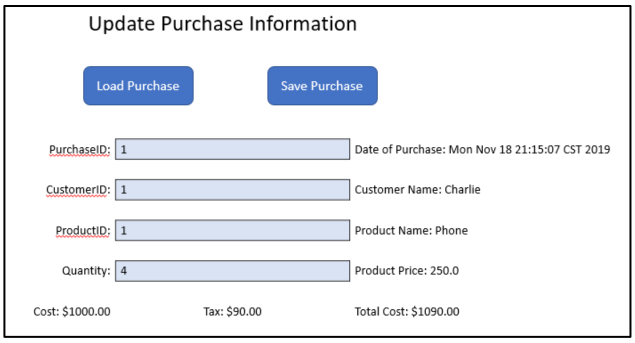
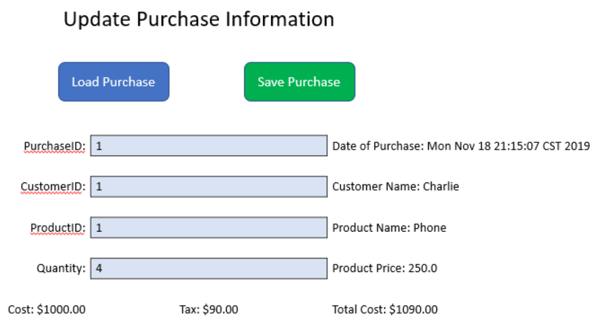
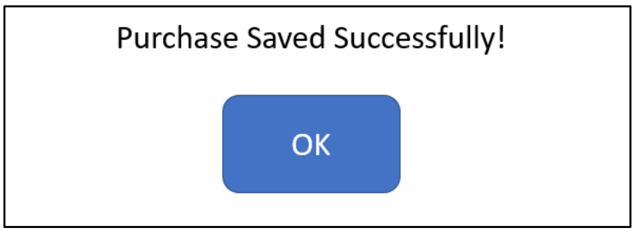
As a user, I want to update a current customer in the system.

Actor	**System**
1. Choose “Manage Customers” on main screen  Store Management System 	2. Display “Update Customer” screen  Update Customer Information CustomerID: Name: Phone: Address:
3. Enter CustomerID for existing customer and click Load Customer  Update Product Information ProductID: 1 Name: Price: Quantity:	4. Display Existing Customer Information  Update Customer Information CustomerID: 1 Name: Charlie Phone: 1234567890 Address: Charlie’s Address
5. Update desired info and click Save Customer  Update Customer Information CustomerID: 1 Name: Charlie Phone: 1234567890 Address: Charlie’s New Address	6. Display “Customer Saved Successfully” screen  Customer Saved Successfully!

As a user, I want to add a new purchase to the system

Actor	**System**
1. Choose “Manage Purchases” on main screen  Store Management System Manage Products · Manage Customers · Manage Purchases	2. Display the Update Purchase screen with the current date stamp  Update Purchase Information Load Purchase · Save Purchase PurchaseID: · Date of Purchase: Mon Nov 18 21:15:07 CST 2019 CustomerID: · Customer Name: ProductID: · Product Name: Quantity: · Product Price: Cost: $0.00 · Tax: $0.00 · Total Cost: $0.00
3. Enter Purchase information.  Update Purchase Information Load Purchase · Save Purchase PurchaseID: 1 · Date of Purchase: Mon Nov 18 21:15:07 CST 2019 CustomerID: 1 · Customer Name: ProductID: 1 · Product Name: Quantity: 3 · Product Price: Cost: $0.00 · Tax: $0.00 · Total Cost: $0.00	4. Autofill info outside of boxes (as user enters Purchase)  Update Purchase Information Load Purchase · Save Purchase PurchaseID: 1 · Date of Purchase: Mon Nov 18 21:15:07 CST 2019 CustomerID: 1 · Customer Name: Charlie ProductID: 1 · Product Name: Phone Quantity: 3 · Product Price: 250.0 Cost: $750.00 · Tax: $67.50 · Total Cost: $817.50
5. Click Save Purchase  Update Purchase Information Load Purchase · Save Purchase PurchaseID: 1 · Date of Purchase: Mon Nov 18 21:15:07 CST 2019 CustomerID: 1 · Customer Name: Charlie ProductID: 1 · Product Name: Phone Quantity: 3 · Product Price: 250.0 Cost: $750.00 · Tax: $67.50 · Total Cost: $817.50	6. Display Purchase Added screen  Purchase Saved Successfully! OK

As a user, I want to update a current purchase in the system.

Actor	**System**
1. Choose “Manage Purchases” on main screen  Store Management System 	2. Display “Update Purchase” screen  Update Purchase Information PurchaseID: Date of Purchase: Mon Nov 18 21:15:07 CST 2019 CustomerID: Customer Name: ProductID: Product Name: Quantity: Product Price: Cost: $0.00 Tax: $0.00 Total Cost: $0.00
3. Enter an existing PurchaseID and click Load Purchase  Update Purchase Information PurchaseID: 1 Date of Purchase: Mon Nov 18 21:15:07 CST 2019 CustomerID: Customer Name: ProductID: Product Name: Quantity: Product Price: Cost: $0.00 Tax: $0.00 Total Cost: $0.00	4. Load all product information  Update Purchase Information PurchaseID: 1 Date of Purchase: Mon Nov 18 21:15:07 CST 2019 CustomerID: 1 Customer Name: Charlie ProductID: 1 Product Name: Phone Quantity: 3 Product Price: 250.0 Cost: $750.00 Tax: $67.50 Total Cost: $817.50
5. Update Purchase information as desired  Update Purchase Information PurchaseID: 1 Date of Purchase: Mon Nov 18 21:15:07 CST 2019 CustomerID: 1 Customer Name: Charlie ProductID: 1 Product Name: Phone Quantity: 4 Product Price: 250.0 Cost: $750.00 Tax: $67.50 Total Cost: $817.50	6. Update the info outside the boxes as the user types  Update Purchase Information PurchaseID: 1 Date of Purchase: Mon Nov 18 21:15:07 CST 2019 CustomerID: 1 Customer Name: Charlie ProductID: 1 Product Name: Phone Quantity: 4 Product Price: 250.0 Cost: $1000.00 Tax: $90.00 Total Cost: $1090.00
7. Click “Save Purchase”  Update Purchase Information PurchaseID: 1 Date of Purchase: Mon Nov 18 21:15:07 CST 2019 CustomerID: 1 Customer Name: Charlie ProductID: 1 Product Name: Phone Quantity: 4 Product Price: 250.0 Cost: $1000.00 Tax: $90.00 Total Cost: $1090.00	8. Display “Purchase Added” screen  Purchase Saved Successfully!

Task 2: Redesign the Data Access layer at the client side that can load/save data to a remote server component.

Prior to this Project, the system worked in a 3-tier system. The User Interface contained the views a user interacted with and entered information into. This passed the information into the Business Logic classes which processed the events of the user screens and created objects which stored the information obtained. Finally, the Data Access tier took the objects created in the Business Logic tier and processed them for containment in a database.

In this project, the Data Access tier of the program was separated so that the database server could be contacted remotely, and by multiple users. The client side of the program must connect to the Data Access tier, which runs separately and in parallel, in order to process the events of the User Interface and store objects in the database.

Task 3: Design the server component to perform load/save requests from the Data Access layer at the client side. Describe the protocol for two sides: client and server

The new 3 tier system is as follows:

1. User Interface
2. Business Logic
3. Data access

Operation begins by running the StoreServer main method. This creates a SQLite data adapter and server socket, as well as a method to respond to interactions with the database server. Then the User Interface is initiated by running the StoreManager main method, which creates a Network Data Adapter which is prepared to save and load products, customers, and purchases from a database. Interacting with the Main UI opens other UIs. Saving an object in any UI creates the corresponding object model and utilizes the save object method from the server data adapter. Loading an object is similar, where an ID is communicated to the server, the Data Access tier searches the database for the object and returns it, and the Business Logic populates the User Interface with the object model.

User Interface:

This is what the user interacts with, and consists of 4 primary views (as outlined above):

- MainUI class
- ManageProductUI class
- ManageCustomerUI class
- ManagePurchaseUI class
- StoreManager class: main class to run the store manager

Business logic:

This is where information entered by a user is interpreted and evaluated in terms of the goals of the store management system. Models for the classes allow for information to be passed into a database:

- ProductModel class: store information of a product
- CustomerModel class: store information of a customer
- PurchaseModel class: store information of a purchase

Data access:

This is where information accumulated in the business logic tier is stored. The tier operates independently and connects with a database, as well as the methods to save and store information in that database.

- StoreServer class: main class for server operation
- IDataAdapter class: contains methods for saving/loading object data from business logic
- NetworkDataAdapter class: implements IDataAdapter, creates socket adapter and message model
- SocketNetworkAdapter
- MessageModel class
- SQLiteAdapter: connect to SQL database
- OracleDataAdapter class: connect to Oracle database (for future implementation)
- INetworkAdapter class

Task 4: Implement the System

The store management system is implemented in Java using the IntelliJ IDE.

Task 5: Test the system with each use case

The use cases are tested in 3 separate videos (one for loading and saving each type of object). Test cases are described in the Project 2 Test Catalog document.

The videos are at the link: https://www.youtube.com/playlist?list=PLxSttXKfgYWTLgee-h7d0Xf4hV8Cpcw-H