Gear Jammers Programmer's Manual

John Ponte
Ryan McAllister-Grum
Tarika Common

Table of Contents

Requirements Documentation

1. Introduction

1.1 Scope

- This project focuses on the creation and integration of an RMA (Return Merchandise Authorization) System to track returns using a graphical interface. The RMA will track returns based on the purchase order # and status of the items being returned. The system will also be able to note specific issues with products including defects, replacements, or refunds for customers. Our RMA will save time and improve efficiency of sales logistics for our target market of merchandising and product companies with warehousing operations.

Requirements
Windows 10
Standalone application
Java and JavaFX

Features
Microsoft SQL database to track customer return order information
Notifications within application for saving and submissions
JavaFX to build GUI using UI Controls

1.2 Definitions/Acronyms

- JDK: Java Development Kit
- MSSQL: Microsoft SQL Server

1.3 References

- We do not reference any trademarked or copyrighted material.

2. General Description

2.1 Product Perspective

- Provides companies with tools to better manage all income returns with a focus on any critical orders. With the ability to now prioritize orders and process returns more efficiently, they can more easily determine whether a product is damaged and needs to be replaced, if a refund needs to be issued or if an item could be repaired. This will allow returns to be properly assessed before transferring to the next appropriate different department.

2.2 Product Functions

- Provides companies with tools to better manage all income returns with a focus on any critical orders. With the ability to now prioritize orders and process returns more efficiently, they can more easily determine whether a product is damaged and needs to be replaced, if a refund needs to be issued or if an item could be repaired. This will allow returns to be properly assessed before transferring to the next appropriate different department.

2.3 User Characteristics

- End user of this product will be warehousing vendors.

2.4 General Constraints

- This software will not work over a network. Windows 10 is required. Microsoft SQL Server is the required database.

2.5 Assumption and dependencies

- We are assuming that the vendor has Java Runtime Environment, they are running Windows 10, have JDK 11 and able to set up database locally based on user documentation.

3. Specific Requirements

1.0 Log-ins

- 1.1 Username
  - 1.1.1 If missing, then Connect is disabled
- 1.2 Password
  - 1.2.1 If missing, then Connect is disabled
- 1.3 User role: analysts, engineers, admins
- 1.4 Instance
  - 1.4.1 If missing, then Connect is disabled
  - 1.4.2 If Username or Password is not filled in, then pop-up error.
- 1.5 Connect Button
  - 1.5.1 Pop-up when connection fails

2.0 Home Screen

- 2.1 Indicator of total items in list with last updated date:time
- 2.2 Table view of all RMAs
  - 2.2.1 List column names in table view
    - 2.2.1.1 RMA, Status Replacement/Repair, RMA# etc (get rest of names)
  - 2.2.2 Row is clickable and a double-click will open RMA Details Screen corresponding to the RMA ID
- 2.3 Search List
  - 2.3.1 If we search, table will update to the RMA# searched
- 2.4 New RMA
  - 2.4.1 Populate/create new RMA – generate new RMA form
  - 2.4.2 Engineers unable to create new RMA requests, button disabled
- 2.5 Refresh Button
  - 2.5.1 Refresh list and grab any new RMAs added
- 2.6 Exit Button
  - 2.6.1 Pop up - "are you sure you want to exit" verification
- 2.7 Checkbox and button delete
  - 2.7.1 Engineers unable to delete RMA requests, button disabled
  - 2.7.2 Delete button does nothing when there are no RMA checked for Delete
  - 2.7.3 Pop up – "Are you sure you want to permanently delete the selected RMAs?" (Confirm Cancel buttons)
  - 2.7.4 Multiple RMA Selections can be deleted

3.0 Database

- 3.1 Product Categories
- 3.2 Products
- 3.3 Customers
- 3.4 Customer Addresses
- 3.5 Purchase Orders
- 3.6 Purchase Order Products
- 3.7 RMA Statuses
- 3.8 RMA

3.9 RMA Details
3.10 Return Reason Code
3.11 Dispositions

4.0 RMA Form

4.1 Customer name drop down list that has all customer names in database
4.1.1 On database connection error, message should pop up reporting error and notifying user they should attempt to retry by selecting the entry again
4.1.2 Connect error will be tracked in customerConnectionError Boolean to allow user to retry if they select the same entry (a different entry will attempt as normal)
4.2 Business name drop down list that has all of the selected customer's business names in databases
4.2.1 On database connection error, message should pop up reporting error and notifying user they should attempt to retry by selecting the entry again
4.2.2 Connection error will be tracked in businessConnectionError Boolean to allow user to retry if they select the same entry (a different entry will attempt as normal)
4.3 Reason Code drop down list that is assigned to customers in database
4.4 Credit, replace, repair drop down uses static drop down list
4.4.1 Depending on whether "Credit" is selected, disabled and clear or re-enable controls under "Replacement Information"
4.4.1.1 Ask user first if any of the controls are populated
4.5 RMA status in database
4.5.1 "Closed" status removed
4.6 RMA Owner set default and assigned to creator of RMA
4.7 Product Information (show access for only 1 product)
4.7.1 Product reference database for list of product and category
4.7.2 Return Label Tracker TextField
4.7.3 Quantity TextField
4.7.3.1 Field will only accept positive numerical numbers
4.8 Initial Evaluation
4.9 Product Disposition
4.9.1 Disposition generates lists from database
4.9.2 Disposition Notes TextField
4.10 Save, Close, Cancel
4.10.1 Pop up to verify user would like to save, save & close, cancel
4.10.2 Error pop up missing required field displays if save or save & cancel is selected and Customer Name, Business Name, PO Number, ReturnReasonCode,

CreditReplaceRepair, RMA Status, Product or Quantity is empty.

4.10.3 On Cancel Pop warning indicating form will be cleared and closed, and no data will be saved.

4.11 PO Number drop down list that has all the of the selected business's PO numbers in database

4.11.1 Selecting PO number clears entered Product Information and populates Product dropdown

4.11.2 On database connection error, message should pop up reporting error and notifying user they should attempt to retry by selecting the entry again

4.11.3 Connection error will be tracked in poConnectionError Boolean to allow user to retry if they select the same entry (a different entry will attempt as normal)

4.12 Customer Shipping Address TextField

4.12.1 Autogenerated when Business Name is chosen

4.13 Additional Information/Special Instruction TextField

4.14 Replacement/Repair Product Tracking # TextField

4.15 Shipping DatePicker

4.16 Shipped Indicator Checkbox

4.17 Close Button same function as Cancel button

4.18 Order of selection needs to be followed

4.18.1 Customer Name selection enables Business Name

4.18.2 Business Name selection enables PO Number

4.18.3 PO Number enables Product

4.18.4 Changes made to a higher tier clears selected value for all lower tiers and disables the control for tiers lower than the next tier down (Cancel button disables all controls underneath the Customer tier)

5.0 Details

5.1 Progress Bar

5.1.1 Updates based on the completion of criteria of the RMA

5.2 Information

5.2.1 Edit button allow for editing of field

5.2.1.1 Popup to confirm would like to process with making change

5.2.1.2 "Edit" button changes to "Save" button text

5.2.1.3 "Save" button text changes to "Edit" button text

5.2.2 RMA ID - autogenerated on save from RMA form and displays

5.2.3 Last Modified By – autogenerates based on trigger from "Edit" button and subsequent "Save" button change

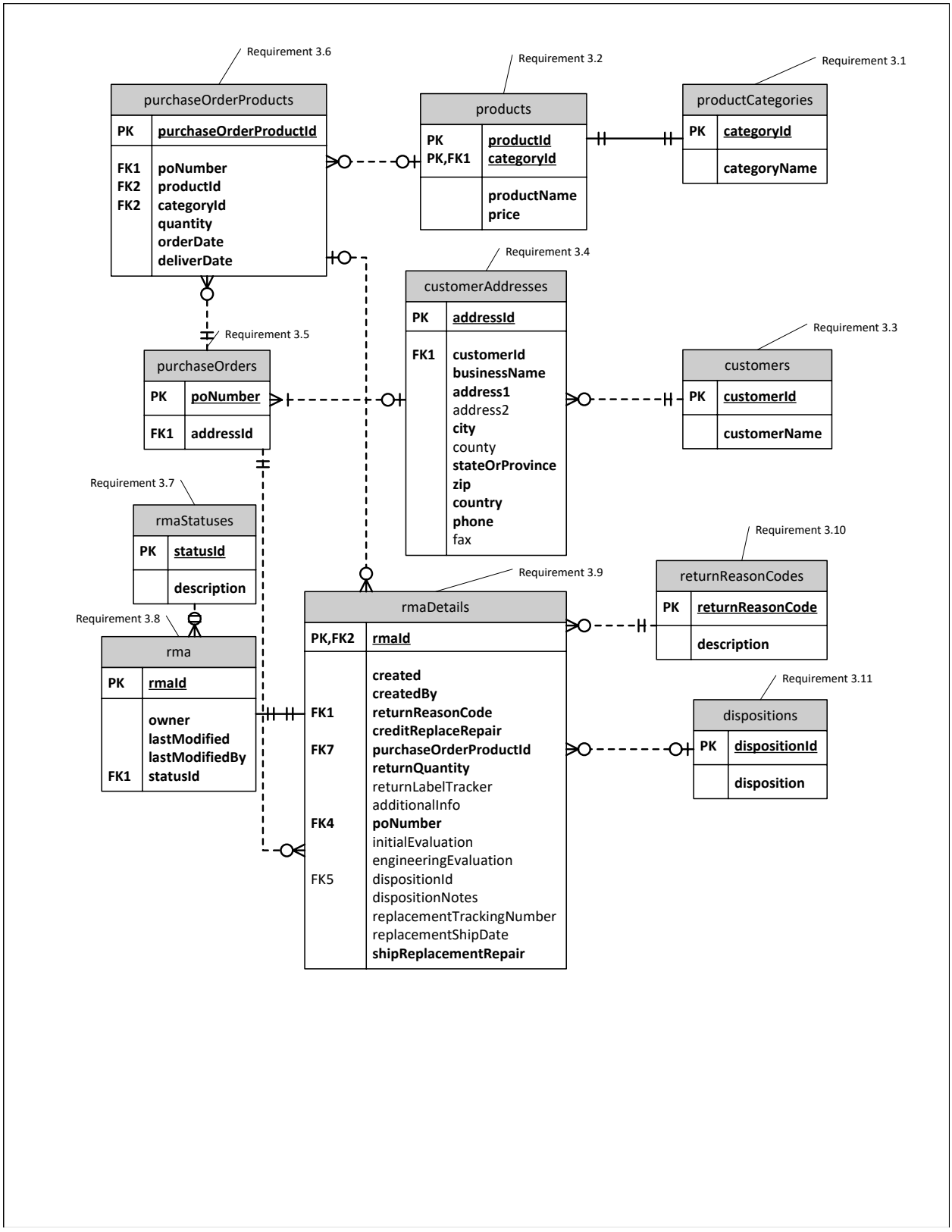
5.2.4 Age in Days – auto-populates on initialization based on RMA creation date and time, or creation and last modified if closed

5.2.5 Created By – auto assigned to the one that submit the New RMA Form

5.2.6 Owner dropdown to allow reassigning RMA to another user
5.2.7 Credit, Replace, Repair dropdown – Depending on whether "Credit" is selected, disabled, clear set values, and collapsed "Replacement Information" section or expand section and re-enable controls
5.2.7.1 Ask user first if any of the controls are populated
5.2.8 EMA Status Closed is available
5.2.8.1 When Status Closed is selected pop up displays
5.2.8.1.1 If RMA is not 100% Error Displays
5.2.8.1.2 If RMA is 100% Confirm Close Displays
5.2.8.2 RMA with Closed status cannot be edited or deleted
5.3 Replacement Details
5.3.1 Button trigger or pop up that says RMA is complete or ready (should removed) close status should have a pop up if RMA is ready to close and can no longer be edited
5.4 Detail Window
5.4.1 Defaulting to the Customer Information that the RMA is linked to
5.4.2 Window trigger hyperlink (currently what we have in place)
5.5 Each hyperlink when clicked, detail window populates info for that particular RMA
5.6 Product Evaluation
5.6.1 Non-Engineers initial evaluation will be enabled
5.6.2 Feature is disabled for Engineers
5.6.2.1 Both fields available or Admins
5.6.3 Engineering Evaluation edit button is disabled for Analysts
5.7 Home Button
5.8 All Comboboxes besides Customer, Business, and PO Number are enabled to be changed and will be updated in the database after every change

Design Documentation

1. Architecture Diagrams



UML Class Diagram

2. Pseudocode

Main

- main(args) {
  - Calls RMA's main method. Used as a workaround to launch the application because attempting to launch using a class that extends Application fails.

  }

RMA

- start(stage) {
  - Load and display the Login form to the user.

  }

- main(args) {
  - Calls the Application.launch(String…) method.

  }

DBConnection

- DBConnection(username, password, instance) throws exception on error {
  - Constructor that takes in a username, password, and instance name, and attempts to create a connection to the database.
  - Create the full URL to the SQL Server database using JDBC that includes username, password, and instance
  - Connect to database

  }

- boolean isConnected() {
  - isConnected checks that we have a connection to the database. Return true if the connection exists, otherwise false.
  - Return true if the connection is not closed and either statement, prepared statement, or callable statement are not closed
    - On exception, return false

  }

DBService

- DBService(username, password, instanceName) throws exception on error {
  - Constructor that takes in a username, password, and instance name, and attempts to create a connection to the database.
  - Call super method (DBConnection's constructor)
  - Store username
  - Get user role using getDBRole() and then store the result

  }

- getInstance(username, password, instanceName) throws exception on error {
  - getInstance is used to set up the static DBService instance and then return it.
  - If (instance is null)
    - Create it using DBService(username, password, instanceName)
  - Return instance

  }

- String getDBRole() throws exception on error {
  - getDBRole returns the logged-in user's assigned role in the database.
  - Create query to fetch this user's role in the RMA database
  - Connect to database
  - Execute query
  - Initialize result as an empty String
  - If (record exists)
    - Assign role to result
  - Close connection
  - Return result

  }

- int getCustomerId(customerName) throws exception on error {
  - getCustomerId fetches the customer ID associated with the passed-in customer name.
  - Create query to fetch the customer ID for the given customerName
  - Connect to database
  - Execute query
  - Initialize result as an integer, set to -1 to signal an error in case we cannot find the ID
  - If (record exists)
    - Assign customer ID to result
  - Close connection
  - Return result

  }

- List<String> getCustomerNames() throws exception on error {

- getCustomerNames returns a list of customer names stored in the database.
- Create query to fetch list of customer names
- Connect to database
- Execute query
- Initialize result to be an empty list of Strings
- While (there are more records to process)
  - Add customer name to result
- Close connection
- Return result

}

- List<CustomerAddress> getCustomerBusinessNames(customerName) throws exception on error {
  - getCustomerBusinessNames returns the list of business names associated with a customer name through a list of CustomerAddress with showBusinessName explicitly set to true.
  - Create query to fetch list of CustomerAddress details associated with customerName
  - Connect to database
  - Execute query
  - Initialize result to be an empty list of CustomerAddress
  - While (there are more records to process) {
    - Initialize a new CustomerAddress
    - Set new CustomerAddress's showBusinessName to true
    - Add CustomerAddress to result
  - }
  - Close connection
  - Return result

}

- CustomerAddress getCustomerAddress(addressId) throws exception on error {
  - getCustomerAddress returns a CustomerAddress containing the address details for the given addressId.
  - Create query to fetch address details for the given address ID
  - Connect to database
  - Execute query
  - Initialize result to be an instance of CustomerAddress, set to null
  - If (record exists)
    - Create a new CustomerAddress using the database details and assign it to result
  - Close connection
  - Return result

}

- List<CustomerAddress> getCustomerAddresses(customerName) throws exception on error {
  - getCustomerAddresses returns a list of CustomerAddress instances containing the details of each address associated with the given customer name.
  - Create query to fetch the details of each address associated with the given customerName
  - Connect to database
  - Execute query
  - Initialize result to be an empty list for instances of CustomerAddress
  - While (there are more records to process)
    - Create a new CustomerAddress using the record's details and add it to result
  - Close connection
  - Return result
  }

- List<String> getCustomerAddressPONumbers(CustomerAddress) throws exception on error {
  - getCustomerAddressPONumbers returns a list of PO numbers associated with the given CustomerAddress.
  - Create query to fetch the list of Purchase Order numbers associated with the given CustomerAddress
  - Connect to database
  - Execute query
  - Initialize result to be an empty list of Strings
  - While (there are more records to process)
    - Add PO number to result
  - Close connection
  - Return result
  }

- PurchaseOrderProduct getPurchaseOrderProduct(purchaseOrderProductId) throws exception on error {
  - getPurchaseOrderProduct returns the requested PurchaseOrderProduct whose id is the passed-in purchaseOrderProductId.
  - Create query to fetch PurchaseOrderProduct details
  - Connect to database
  - Execute query
  - Initialize result to be an instance of PurchaseOrderProduct, set to null
  - If (record exists)
    - Create a new PurchaseOrderProduct using the database details and assign it to result
  - Close connection
  - Return result}

- List<PurchaseOrderProduct> getPurchaseOrderProducts(poNumber) throws exception on error {
  - getPurchaseOrderProducts returns a list of PurchaseOrderProducts associated with the given PO number.
  - Create query to fetch purchase order product details associated with a given purchase order number
  - Connect to database
  - Execute query
  - Initialize result to be an empty list for instances of PurchaseOrderProduct
  - While (there are more records to process)
    - Create a new PurchaseOrderProduct using the database details and add it to result
  - Close connection
  - Return result

  }

- int getRMAStatusId(description) throws exception on error {
  - getRMAStatusId returns the statusId associated with the passed-in status description.
  - Create query to fetch the status ID for the given description
  - Connect to database
  - Execute query
  - Initialize result as an integer, set to -1 to signal an error in case we cannot find the description
  - If (record exists)
    - Assign id to result
  - Close connection
  - Return result

  }

- String getRMAStatusDescription(statusId) throws exception on error {
  - getRMAStatusDescription returns the description associated with an RMA status ID.
  - Create query to fetch the description for the associated RMA Status ID
  - Connect to database
  - Execute query
  - Initialize result as an empty String
  - If (record exists)
    - Assign description to result
  - Close connection
  - Return result

  }

- List<String> getRMAStatuses() throws exception on error {
  - getRMAStatuses returns a list of Strings containing the RMA status descriptions in the database.
  - Create query to fetch the RMA status descriptions
  - Connect to database
  - Execute query
  - Initialize result as an empty list of Strings
  - while (there are more records to process)
    - Add RMA status description to result
  - Close connection
  - Return result

  }

- Map<String, String> getReturnReasonCodes() throws exception on error {
  - getReturnReasonCodes returns a Map containing types String, whose contents are the return reason code initials and the associated description.
  - Create query to fetch the return reason code and description pairs from the database
  - Connect to database
  - Execute query
  - Initialize result as an empty Map to store return reason code, description pairs
  - While (there are more records to process)
    - Put the return reason code and its description into result
  - Close connection
  - Return result

  }

- int getDispositionId(disposition) throws exception on error {
  - getDispositionId returns the int identifier for the passed-in disposition description.
  - Create query to select the disposition ID for the given disposition description
  - Connect to database
  - Execute query
  - Initialize result as an integer, set to -1 to signal an error in case we cannot find the description
  - If (record exists)
    - Assign disposition id to result
  - Close connection
  - Return result

  }

- List<String> getDispositions() throws exception on error {
  - getDispositions returns a List containing the list of available disposition descriptions in the database.
  - Create query to fetch to fetch the disposition descriptions in the database
  - Connect to database
  - Execute query
  - Initialize result to be an empty list of Strings
  - While (there are more records to process)
    - Add description to result
  - Close connection
  - Return result
  }

- String createRMA(statusDescription) throws exception on error {
  - createRMA creates the initial RMA record in the database using the passed-in status description and returns the generated RMA ID that was used to create the record.
  - Create callable query to execute the GetNextRMAId stored procedure that calculates the next RMA ID
  - Declare and set output variable for callable statement
  - Execute callable statement
  - Fetch the next RMA ID
  - Close connection
  - Fetch the RMA Status ID for the passed-in description using getRMAStatusId(statusDescription)
  - Create prepared statement to insert the new RMA record
  - Populate prepared statement with the next RMA ID, the fetched RMA Status ID, the current username, and current date and time
  - Execute prepared statement
  - Close connection
  - Return the RMA ID
  }

- RMAListViewModel getRMA(rmaId) throws exception on error {
  - getRMA fetches the details for the given RMA ID and returns it as an RMAListViewModel.
  - Create query to fetch the RMA details used to populate an RMAListViewModel, which includes creating a CustomerAddress and PurchaseOrderProduct
  - Connect to database
  - Execute query
  - Initialize result to be an instance of RMAListViewModel, set to null
  - If (record exists)
    - Create a new RMAListViewModel instance and assign it to result
  - Close connection
  - Return result }

- List<RMAListViewModel> getRMAs() throws exception on error {
  - getRMAs fetches the list of open RMAs in the database. If the user is an engineer, then the list of RMAs will be all those with an empty engineeringEvaluation in RMADetails. In both cases, if a critical RMA exists (RMA that has not been modified for five or more days), then the list will only show critical RMAs.
  - Initialize result to be an empty list for instances of RMAListViewModel
  - Create initial query to fetch critical RMAs depending on whether the user is an Engineer based on an unspecified end date and time
  - Connect to database
  - Create prepared statement using query
  - Populate prepared statement with the current date and time
  - Execute prepared statement
  - If (record exists)
    - do
      - Create RMAListViewModel using details and add to result
    - while (there are more records to process)
  - else
    - Close connection so we can try again, this time checking for non-critical RMAs using an unspecified end date and time
    - Connect to database
    - Create prepared statement using query
    - Populate prepared statement with the current date and time
    - Execute prepared statement
    - if (record exists)
      - do
        - Create RMAListViewModel using details and add to result
      - while (there are more records to process)
  - Close connection
  - Return result

  }

- createRMADetails(rmaId, returnReasonCode, creditReplaceRepair, purchaseOrderProductId, returnQuantity, returnLabelTracker, additionalInfo, poNumber, initialEvaluation, engineeringEvaluation, disposition, dispositionNotes, replacementTrackingNumber, replacementShipDate, shipReplacementRepair) throws exception on error {
  - createRMADetails creates the details record associated with the initial RMA record using the passed-in information.
  - Create query to fetch the dispositionId associated with the passed-in disposition
  - Connect to database
  - Execute query
  - declare dispositionId to hold result
  - If (record exists)
    - Fetch disposition ID and assign to dispositionId
  - else
    - throw exception indicating that we are unable to find the description
  - Create insert statement to insert the new record into RMA details
  - Create prepared statement using query
  - Populate prepared statement with rmaId, returnReasonCode, creditReplaceRepair, purchaseOrderProductId, returnQuantity, returnLabelTracker, additionalInfo, poNumber, initialEvaluation, engineeringEvaluation, disposition, dispositionNotes, replacementTrackingNumber, replacementShipDate, and shipReplacementRepair
  - Execute prepared statement
  - Close connection

  }

- String getRMACustomerName(rmaId) throws exception on error {
  - getRMACustomerName returns the customer name associated with an RMA ID, through its PO number and the PO number's associated address ID.
  - Create query to fetch the associated with the RMA ID
  - Connect to database
  - Execute query
  - Initialize result as an empty String
  - If (record exists)
    - Assign customer name to result
  - Close connection
  - Return result

  }

- CustomerAddress getRMABusinessName(rmaId) throws exception on error
  {
  - getRMABusinessName returns a CustomerAddress set to show the business name associated with the RMA whose ID we are searching, through its PO number and the PO number's associated address ID.
  - Create query to fetch business name
  - Connect to database
  - Initialize result to be an instance of CustomerAddress, set to null
  - If (record exists) {
    - Instantiate result with a new CustomerAddress containing the record details
    - Set result's showBusinessName to true
    - }
  - Close connection
  - Return result

  }

- String getRMAPONumber(rmaId) throws exception on error {
  - getRMAPONumber returns the PO number associated with the given RMA ID.
  - Create query to fetch PO number
  - Connect to database
  - Initialize result as an empty String
  - If (record exists)
    - Assign PO number to result
  - Close connection
  - Return result

  }

- CustomerAddress getRMAAddress(rmaId) throws exception on error {
  - getRMAAddress returns the address information associated with the RMA ID's PO number in a CustomerAddress.
  - Create query to fetch address details necessary to create a CustomerAddress based on RMA ID
  - Connect to database
  - Initialize result as an instance of CustomerAddress, set to null
  - if (record exists)
    - Use database details to create a new CustomerAddress and assign it to result
  - Close connection
  - Return result

  }

- List<String> getOwners() throws exception on error {
  - getOwners returns a list of usernames stored in the RMA database to be used as potential RMA owners.
  - Create query to fetch usernames
  - Connect to database
  - Execute query
  - Initialize result as an empty list of Strings
  - while (there are more records to process)
    - Add username to result
  - Close connection
  - Return result

  }

- String getRMAOwner(rmaId) throws exception on error {
  - getRMAOwner returns the username of the owner stored in the database for the specified RMA ID.
  - Create query to fetch owner
  - Connect to database
  - Execute query
  - Initialize result as an empty String
  - If (record exists)
    - Assign username to result
  - Close connection
  - Return result

  }

- updateRMAOwner(rmaId, newOwner) throws exception on error {
  - updateRMAOwner updates the RMA owner in the database for the given RMA ID with the passed-in username.
  - Create query to update owner in database for a given RMA ID
  - Create prepared statement using query
  - Populate prepared statement with rmaId and newOwner
  - Execute prepared statement
  - Close connection

  }

- LocalDateTime getRMALastModified(rmaId) throws exception on error {
  - getRMALastModified fetches the datetime stored in the database for the given RMA ID.
  - Create query to fetch the last modified datetime for the given RMA ID
  - Connect to database
  - Execute query
  - Initialize result as an instance of LocalDateTime, set to null
  - If (record exists)
    - Assign last modified datetime to result
  - Close connection

- Return result
}

- updateRMALastModified(rmaId) throws exception on error {
  - updateRMALastModified updates the date and time for the given RMA ID's lastModified column to now.
  - Create query to update lastModified based on a given RMA ID
  - Create prepared statement using query
  - Populate prepared statement with rmaId and the current date and time
  - Execute prepared statement
  - Close connection
  }

- String getRMALastModifiedBy(rmaId) throws exception on error {
  - getRMALastModifiedBy returns the username of the user that last modified the RMA with the given ID.
  - Create query to fetch the lastModifiedBy username for the given RMA ID
  - Connect to database
  - Execute query
  - Initialize result as an empty String
  - if (record exists)
    - Assign lastModifiedBy username to result
  - Close connection
  - Return result
  }

- updateRMALastModifiedBy(rmaId) throws exception on error {
  - updateRMALastModifiedBy updates the username in the lastModifiedBy column of the RMA with the given ID to the current logged-in user.
  - Create query to update lastModifiedBy based a given RMA ID
  - Connect to database
  - Create prepared statement using the query
  - Populate the prepared statement with rmaId and lastModifiedBy
  - Execute prepared statement
  - Close connection
  }

- String getRMAStatus(rmaId) throws exception on error {
  - getRMAStatus returns the status description stored in the database for the given RMA ID.
  - Create query to fetch the RMA status description for the given RMA ID
  - Connect to database
  - Execute query
  - Initialize result as an empty String
  - If (record exists)
    - Assign RMA status description to result
  - Close connection
  - Return result

  }

- updateRMAStatus(rmaId, statusDescription) throws exception on error {
  - updateRMAStatus updates the status assigned to the RMA with the given ID to the passed-in status description.
  - Create query to update RMA status for a given RMA ID using a given status description
  - Connect to database
  - Create prepared statement using query
  - Populate prepared statement with rmaId and statusDescription
  - Execute prepared statement
  - Close connection

  }

- String getRMACreditReplaceRepair(rmaId) throws exception on error {
  - getRMACreditReplaceRepair returns the current value (credit, replace, or repair) stored in the database for the given RMA ID.
  - Create query to fetch the current creditReplaceRepair value for the given RMA ID
  - Connect to database
  - Execute query
  - Initialize result as an empty String
  - if (record exists)
    - Assign value to result
  - Close connection
  - Return result

  }

- updateRMACreditReplaceRepair(rmaId, creditReplaceRepair) throws exception on error {
  - updateRMACreditReplaceRepair updates the specified RMA with the passed-in credit, replace, or repair value.
  - Create query to update the creditReplaceRepair column of an RMA detail record for a given RMA ID with the given creditReplaceRepair value
  - Connect to database
  - Create prepared statement using the query
  - Populate the prepared statement with rmaId and creditReplaceRepair
  - Execute prepared statement
  - Close connection

  }

- String getRMAReturnReasonCode(rmaId) throws exception on error {
  - getRMAReturnReasonCode returns the current return reason code description associated with the given RMA ID.
  - Create query to fetch the return reason code for the given RMA ID
  - Connect to database
  - Execute query
  - Initialize result as an empty String
  - if (record exists)
    - Assign the return reason code description to result
  - Close connection
  - Return result

  }

- updateRMAReturnReasonCode(rmaId, returnReasonCodeDescription) throws exception on error {
  - updateRMAReturnReasonCode updates the specified RMA with the passed-in Return Reason Code description.
  - Create query to update the RMA details for a given RMA ID with a given Return Reason Code description
  - Connect to database
  - Create prepared statement using query
  - Populate prepared statement with rmaId and returnReasonCodeDescription
  - Execute prepared statement
  - Close connection

  }

- String getRMAAdditionalInfo(rmaId) throws exception on error {
  - getRMAAdditionalInfo returns the additional info notes stored in the RMA Details for the given RMA ID.
  - Create query to fetch the additionalInfo for the given RMA ID
  - Connect to database
  - Execute query

- Initialize result as an empty String
- If (record exists)
  - Assign additional info to result
- Close connection
- Return result

}

- updateRMAAdditionalInfo(rmaId, additionalInfo) throws exception on error {
  - updateRMAAdditionalInfo updates the requested RMA's details with the updated information passed-in through additionalInfo.
  - Create query to update the additional information in RMA details for a given RMA ID
  - Connect to database
  - Create prepared statement using the query
  - Populate prepared statement with rmaId and additionalInfo
  - Execute prepared statement
  - Close connection

}

- LocalDateTime getRMACreated(rmaId) throws exception on error {
  - getRMACreated returns a LocalDateTime object containing the creation date and time of the RMA with the given ID.
  - Create query to fetch the created date and time for the given RMA ID
  - Connect to database
  - Execute query
  - Initialize result to be an instance of LocalDateTime, set to null
  - If (record exists)
    - Assign the created datetime info to result
  - Close connection
  - Return result

}

- String getRMACreatedBy(rmaId) throws exception on error {
  - getRMACreatedBy returns the username that created the given RMA request.
  - Create query to fetch the created by details for the given RMA ID
  - Connect to database
  - Execute query
  - Initialize result as an empty String
  - If (record exists)
    - Assign created by username to result
  - Close connection
  - Return result

}

- PurchaseOrderProduct getRMAProduct(rmaId) throws exception on error
  {
  - getRMAProduct returns a PurchaseOrderProduct containing the details of the product that is being returned in the given RMA.
  - Create query to fetch the purchase order product details for the given RMA ID
  - Connect to database
  - Execute query
  - Initialize result to be an instance of PurchaseOrderProduct, set to null
  - if (record exists)
    - Create a new PurchaseOrderProduct using the database details and assign it to result
  - Close connection
  - Return result

  }

- updateRMAProduct(rmaId, purchaseOrderProductId) throws exception on error {
  - updateRMAProduct updates the product being returned in the RMA with the passed-in integer purchaseOrderProductId.
  - Create query to update the purchase order product id referenced by a given RMA ID
  - Connect to database
  - Create prepared statement using the query
  - Populate prepared statement with rmaId and purchaseOrderProductId
  - Execute prepared statement
  - Close connection

  }

- int getRMAReturnQuantity(rmaId) throws exception on error {
  - getRMAReturnQuantity fetches the integer number of products being returned by the customer for the given RMA ID.
  - Create query to fetch the return quantity for the given RMA ID
  - Connect to database
  - Execute query
  - Initialize result to be an integer containing -1 in case we are unable to find the record
  - If (record exists)
    - Assign return quantity to result
  - Close connection
  - Return result

  }

- updateRMAReturnQuantity(rmaId, returnQuantity) throws exception on error {
  - updateRMAReturnQuantity updates the int value for the amount of product being returned by the customer for the given RMA ID.
  - Create query to update the return quantity for a given RMA ID
  - Connect to database
  - Create prepared statement from query
  - Populate prepared statement with rmaId and returnQuantity
  - Execute prepared statement
  - Close connection

  }

- String getRMAReturnLabelTracker(rmaId) throws exception on error {
  - getRMAReturnLabelTracker fetches the return label tracking number stored in the RMA with the referenced ID.
  - Create query to fetch return label tracking number
  - Connect to database
  - Execute query
  - Initialize result as an empty String
  - if (record exists)
    - Assign return label tracking number to result
  - Close connection
  - Return result

  }

- updateRMAReturnLabelTracker(rmaId, returnLabelTracker) throws exception on error {
  - updateRMAReturnLabelTracker updates the requested RMA's details with the new passed-in return label tracking ID.
  - Create query to update the return label tracking number for a given RMA ID
  - Connect to database
  - Create prepared statement from query
  - Populate prepared statement with rmaId and returnLabelTracker
  - Execute prepared statement
  - Close connection

  }

- String getRMAInitialEvaluation(rmaId) throws exception on error {
  - getRMAInitialEvaluation fetches the initial evaluation done by an Analyst for the requested RMA.
  - Create query to fetch the initial evaluation for the provided RMA ID
  - Connect to database
  - Execute query
  - Initialize result as an empty String
  - If (record exists)
    - Assign the initial evaluation stored in the database to result

- Close connection
- Return result
}

- updateRMAInitialEvaluation(rmaId, initialEvaluation) throws exception on error {
  - updateRMAInitialEvaluation updates the requested RMA's initial evaluation from an Analyst with the new passed-in initial evaluation text.
  - Create query to update the initial evaluation to a given value for a given RMA ID
  - Connect to database
  - Create prepared statement from query
  - Populate prepared statement with rmaId and initialEvaluation
  - Execute prepared statement
  - Close connection
  }

- String getRMAEngineeringEvaluation(rmaId) throws exception on error {
  - getRMAEngineeringEvaluation fetches the engineering evaluation done by an Engineer for the requested RMA.
  - Create query to fetch the engineering evaluation for the requested RMA ID
  - Connect to database
  - Execute query
  - Initialize result as an empty String
  - if (record exists)
    - Assign engineering evaluation in database to result
  - Close connection
  - Return result
  }

- updateRMAEngineeringEvaluation(rmaId, engineeringEvaluation) throws exception on error {
  - updateRMAEngineeringEvaluation updates the requested RMA's engineering evaluation from an Engineer with the new passed-in engineering evaluation text.
  - Create query to update the engineering evaluation for a given RMA ID
  - Connect to database
  - Create prepared statement using query
  - Populate prepared statement with rmaId and engineeringEvaluation
  - Execute prepared statement
  - Close connection
  }

- String getRMADisposition(rmaId) throws exception on error {
  - getRMADisposition fetches the disposition for the requested RMA.
  - Create query to fetch the disposition description for the given disposition ID assigned to the specific RMA ID
  - Connect to database
  - Execute query
  - Initialize result as an empty String
  - if (record exists)
    - Assign disposition description to result
  - Close connection
  - Return result

  }

- updateRMADisposition(rmaId, disposition) throws exception on error {
  - updateRMADisposition updates the disposition ID stored in the specified RMA using the passed-in disposition description.
  - Create query to update the disposition ID for a given RMA using a given disposition
  - Connect to database
  - Create prepared statement from query
  - Populate prepared statement with rmaId and disposition
  - Execute prepared statement
  - Close connection

  }

- String getRMADispositionNotes(rmaId) throws exception on error {
  - getRMADispositionNotes fetches the disposition notes for the requested RMA.
  - Create query to fetch disposition notes for the given RMA ID
  - Connect to database
  - Execute query
  - Initialize result as an empty String
  - if (record exists)
    - Assign disposition notes to result
  - Close connection
  - Return result

  }

- updateRMADispositionNotes(rmaId, dispositionNotes) throws exception on error {
  - updateRMADispositionNotes updates the disposition notes stored in the specified RMA using the passed-in dispositionNotes.
  - Create query to update disposition notes for a given RMA ID
  - Connect to database
  - Create prepared statement using query
  - Populate prepared statement with rmaId and dispositionNotes
  - Execute prepared statement
  - Close connection }

- String getRMAReplacementTrackingNumber(rmaId) throws exception on error {
  - getRMAReplacementTrackingNumber fetches the tracking number that will be used to ship the replacement back to the customer for the requested RMA.
  - Create query to fetch replacement tracking number for the given RMA ID
  - Connect to database
  - Execute query
  - Initialize result to an empty String
  - if (record exists)
    - Assign replacement tracking number to result
  - Close connection
  - Return result

  }

- updateRMAReplacementTrackingNumber(rmaId, replacementTrackingNumber) throws exception on error {
  - updateRMAReplacementTrackingNumber updates the specified RMA's tracking number used to ship the replacement product.
  - Create query to update the replacement tracking number for a specified RMA ID
  - Connect to database
  - Create prepared statement from query
  - Populate prepared statement with rmaId and replacementTrackingNumber
  - Execute prepared statement
  - Close connection

  }

- LocalDate getRMAReplacementShipDate(rmaId) throws exception on error {
  - getRMAReplacementShipDate fetches the date that the replacement will be shipped to the customer for the given RMA.
  - Create query to fetch the replacement ship date for the given RMA ID
  - Connect to database
  - Execute query
  - Initialize result to be an instance of LocalDate, set to null
  - if (record exists)
    - Assign the replacement ship date to result
  - Close connection
  - Return result

  }

- updateRMAReplacementShipDate(rmaId, replacementShipDate) throws exception on error {
  - updateRMAReplacementShipDate updates the date the replacement will be shipped to the customer with the passed-in date in the RMA with the specified ID.
  - Create query to update the replacement ship date with a specified date for a specified RMA ID
  - Connect to database
  - Create prepared statement using query
  - Populate prepared statement with rmaId and replacementShipDate
  - Execute prepared statement
  - Close connection
  }

- boolean getRMAShipReplacementRepair(rmaId) throws exception on error {
  - getRMAShipReplacementRepair fetches the boolean value of whether we will be shipping a replacement or repair to the customer referenced in the RMA with the given ID.
  - Create query to fetch the set value for shipping the replacement or repair for the given RMA ID
  - Connect to database
  - Execute query
  - Initialize result as a boolean, set to false
  - if (record exists)
    - Assign the set value for shipReplacementRepair to result
  - Close connection
  - Return result
  }

- updateRMAShipReplacementRepair(rmaId, shipReplacementRepair) throws exception on error {
  - updateRMAShipReplacementRepair updates the RMA with the given ID with the new boolean value indicating whether we will be shipping a replacement or repair to the customer.
  - Create query to update the shipReplacementRepair bit value for a specific RMA ID
  - Connect to database
  - Create prepared statement using query
  - Populate prepared statement using rmaId, shipReplacementRepair
  - Execute prepared statement
  - Close connection
  }

- deleteRMA(rmaId) throws exception on error {
  - deleteRMA deletes the RMA with the given ID.
  - Create query to delete the RMA record with a given ID
  - Connect to database

- Create prepared statement from query
- Populate prepared statement using rmaId
- Execute query
- Close connection

}

LoginController

- initialize(url, rb) {
  - initialize ties together the GUI's elements with the LoginModel model and creates any necessary ChangeListeners on the controls.
  - Initialize a new LoginModel
  - Bind username, password, and instances between form and model
  - If Username changes(new ChangeListener<String>() {
    - @Override
    - changed(observableValue, oldValue, newValue) {
      - If (Instance value is not null)
        - If (Password text is not empty and Instances is not empty)
          - Enable Connect button if newValue is not empty or blank
        - Else
          - Disable Connect button
    - }
  - }
  - If Password changes(new ChangeListener<String>() {
    - @Override
    - changed(observableValue, oldValue, newValue) {
      - If (Instance value is not null)
        - If (Username text is not empty and Instances is not empty)
          - Enable Connect button if newValue is not empty or blank
        - Else
          - Disable Connect button
    - }
  - }
  - If Instances value changes(new ChangeListener<String>() {
    - @Override
    - changed(observableValue, oldValue, newValue) {
      - Set Connect button's disabled status depending on if newValue is empty or blank
    - }
  - }

}

- chobInstancesOnAction(event) {
  - Check if the username field is null or empty, and display an Alert if so
  - Check if the password field is null or empty, and display an Alert if so

  }

- btnConnectOnAction(event) {
  - handleConnect first checks that the user has filled out every field, and then attempts to connect to the RMA database stored in the user-selected SQL Server instance. If successful, it initializes an instance of DBService, and then launches the main RMAListView window. If it is unable to connect or find an RMA database in the given instance, a pop-up is shown.
  - First disable the connect Button to prevent repeated clicking while the program is working
  - If (the username, password, and instances ChoiceBox are not empty or blank)
    - try {
      - Initialize DBService using the user-entered username, password, and instance name
      - Load the RMAListViewModel screen
      - Hide the Login screen
    - } catch (SQLException) {
      - Notify user that there was an error when attempting to connect to the SQL Server database, and show them the error message
    - } catch (IOException) {
      - Notify the user that there was an error while attempting to load the RMAListView form, and show them the error message
    - }
  - Re-enable the connect button

  }

LoginModel

- getInstances() {
  - getInstances issues a request to any listening SQL Server Browser Service on default UDP port 1434 for their information of running instances, and any that respond are added to the list of instances in the format HOSTNAME\INSTANCE_NAME. If no hosts respond, an error message is shown and the program exits.
  - Create a DataGram socket in order to send UDP packets
  - try {
    - Create a DataGram packet to send a single-byte array with the value 2 to default UDP address 1434 of the SQL Browser Service to address 255.255.255.255 in order to search for all listening services on the network
    - Send packet
    - Create byte array buffer to store received contents

        - Receive contents
        - Split response on ";" into array of String tokens
        - Create host and instance String variables
        - For (each token) {
            - If (token value equals "ServerName")
                - Set host variable to the next token value
            - Else if (token value equals "InstanceName") {
                - Set instance variable to the next token's value
                - Add host and instance variables to instances Observable Array List
            - }
        - }
        - Close socket
    - } catch (IOException) {
        - Notify user that no SQL Server Browser Services were responding along with the error message and exit the program
    - }
    }

RMAListViewController

- initialize(url, rb) {
    - Initializes the GUI's controls and fetches open RMAs using RMAListViewModels to back the table.
    - Initialize Observable List that will back a Filtered List that will back the table
    - Set textProperty of the item count label
    - Add changeListener to the search text field to update the filtered list with the new value as appropriate so that it will filter the table, and also update the resulting list count
    - Setup the cell value factory for each column so that they display their results properly
    - Add mouse press listener to table to listen for double-clicks to handle opening RMA in RMA Details
    - Fetch the list of open RMAs by calling getRMAList()
    }

- getRMAList() {
    - Fetches the list of open RMAs in the database.
    - Clear the Observable List that is backing the Filtered List that is backing the table
    - try {
        - Fetch the list of open RMAs using DBService's getRMAs() function
    - } catch (SQLException) {
        - Notify user that error has occurred while fetching the list of RMAs
    - } finally {

        - Update the table count with the new resulting count and the current date and time
    - }
    }

- btnNewRMAOnAction(event) {
    - Launches the New RMA form.
    - try {
        - Create a new instance of the New RMA form and show it to the user modally
        - Refresh the list
    - } catch (IOException) {
        - Notify user that there was an error while attempting to create the New RMA form
    - } catch (SQLException) {
        - Notify the user that there was an issue connecting to the SQL Server database
    - }
    }

- btnDeleteOnAction(event) {
    - Deletes any RMAs selected by the user.
    - Create List to hold the RMAListViewModels to be deleted
    - Try {
        - For (each RMAListViewModel in the table)
            - If (it is flagged for deletion from the user checking its box in the "Should Delete?" column) {
                - If (the user has not confirmed deleting the RMAs) {
                - Ask user if they are sure they want to delete the RMAs
                - If (yes)
                - Set confirmDelete to true
                - Else
                - break
                - }
                - Delete the RMA from the database
                - Add RMA to deletion List
            - }
    - } catch (SQLException) {
        - Notify user of error
    - } finally {
        - Remove RMAs added to List from table
        - Refresh table
    - }
    }

RMAListViewModel

- RMAListViewModel(rmaId) throws exception on error {
  - Constructor for RMAListViewModel that uses an RMA ID to fetch the necessary information.
  - Create model containing relevant data using DBService's getRMA(rmaId) method
  - Assign fields using above model
  }

CustomerAddress

- CustomerAddress(addressId) throws exception on error {
  - Constructor for CustomerAddress that takes the passed-in addressId and fetches the address information from the database.
  - Initialize a CustomerAddress to hold the results from calling DBService's getCustomerAddress(addressId) method
  - Assign fields using the result from getCustomerAddress
  }

PurchaseOrderProduct

- PurchaseOrderProduct(purchaseOrderProductId) throws exception on error {
  - Constructor that takes the passed-in purchaseOrderProductId and fetches the product's information from the database.
  - Initialize a PurchaseOrderProduct to hold the results from calling DBService's getPurchaseOrderProduct(purchaseOrderProductId) method
  - Assign fields using the result from getPurchaseOrderProduct
  }

RMAFormController

- Initialize(url, rb){
  - This will initialize elements in the U and set up ChangeListeners for specific controls.
  - Set model to a new RMAFormModel
  - Create a TextFormatter to restrict input into Quantity TextField
  - Setup bindings between model and GUI controls and apply TextFormatter to Quantity
  - Set customerConnectionError, businessConnectionError, and poConnectionError to false
  - When change is made to Customer Name(call new ChangeListener<String>() {
    - @Override
    - public void changed(ObservableValue<? extends String> observableValue, String oldValue, String newValue) {
      - If (newValue is not null and oldValue is not null) {
        - If (newValue does not equal oldValue) {
          - If (customerRevert is false and selected Business Name is not null) {

- Ask if user wants to confirm making their change, as it will clear Business Name, PO Number, Shipping Address, and Product
- If (yes) {
  - Clear out all the values and list in PO Number, and Product, set Shipping Address to EMPTY_ADDRESS, clear the selected Business Name and values, and disable PO Number and Product
  - try {
    - Set the new values for Business Name
    - Set customerConnectionError to false
  - } catch (SQLException) {
    - Notify user of error connecting to database and tell them they can try again
    - Set customerConnectionError to true
  - } finally {
    - Set businessConnectionError to false
  - }
- } else {
  - Set customerRevert to true
  - Set the selected customer name to oldValue
- }

- } else if (customerRevert is false) {
  - Clear Business Name values
  - try {
    - Fetch values for Business Name for the selected Customer Name
    - Set customerConnectionError to false
  - } catch (SQLException) {
    - Notify user of error connecting to database and tell them they can try again
    - Set customerConnectionError to true
  - }
- } else
  - Set customerRevert to true

- }

◊ } else if (newValue is not null) {

- try {
  - Fetch values for Business Name for the selected Customer Name
  - Enable Business Name control
  - Set customerConnectionError to false
- } catch (SQLException) {
  - Notify user of error connecting to database and tell them they can try again
  - Set customerConnectionError to true
- }

```
        ◊  }
    ♦  }
o  }
o  When change is made to Business Name(call new
   ChangeListener<CustomerAddress>() {
    ♦  @Override
    ♦  public void changed(ObservableValue<? extends
       CustomerAddress> observableValue, CustomerAddress oldValue,
       CustomerAddress newValue) {
       ◊  If (newValue is null and oldValue is not null) {
          •  (if the selected Customer Name is null) {
             o  Clear Business Name values
             o  Disable Business Name
             o  Set Shipping Address to EMPTY_ADDRESS
          •  } else {
             o  try {
                ♦  Set selected Business Name to null
                ♦  Fetch Business Names for the given selected
                   Customer Name
                ♦  Enable Business Name
                ♦  Set businessConnectionError to false
                ♦  Set customerConnectionError to false
             o  } catch (SQLException) {
                ♦  Notify user of error connecting to database and
                   tell them they can try again
                ♦  Set businessConnectionError to true
                ♦  Set customerConnectionError to true
             o  }
          •  }
       ◊  } else if (newValue is not null and oldValue is not null)
          {
          •  If (newValue does not equal oldValue) {
             o  If (businessRevert is false and the selected PO Number
                is not empty) {
                ♦  Ask if user wants to confirm making their change,
                   as it will clear PO Number and Product
                ♦  If (yes) {
                   ◊  Clear out the lists and set the selected values
                      to null for PO Number and Product
                   ◊  Set Shipping Address to the newValue's
                      createShippingAddress() output
                   ◊  try {
                      •  Set the new values for PO Number
                      •  Set businessConnectionError to false
                   ◊  } catch (SQLException) {
                      •  Notify user of error connecting to database
                         and tell them they can try again
```

```
                    ●  Set businessConnectionError to true
                ◊  } finally {
                    ●  Set poConnectionError to false
                ◊  }
            ♦  } else {
                ◊  Set businessRevert to true
                ◊  Set selected business name to oldValue
            ♦  }
        o  } else if (businessRevert is false) {
            ♦  Set Shipping Address to newValue's
               createShippingAddress()
            ♦  Clear the list for PO Number
            ♦  try {
                ◊  Fetch values for PO Number for the selected
                   Business Name
                ◊  Set businessConnectionError to false
            ♦  } catch (SQLException) {
                ◊  Notify user of error connecting to database and
                   tell them they can try again
                ◊  Set businessConnectionError to true
            ♦  }
        o  } else
            ♦  Set businessRevert to false
    ●  }
◊  } else if (newValue is not null) {
    ●  Set Shipping Address to newValue's
       createShippingAddress()
    ●  try {
        o  Fetch values for PO Number for the selected Business
           Name
        o  Enable PO Number
        o  Set businessConnectionError to false
    ●  } catch (SQLException) {
        o  Notify user of error connecting to database and tell
           them they can try again
        o  Set businessConnectionError to true
    ●  }
◊  }
♦  }
o  }
o  When change is made to PO Number(call new
   ChangeListener<String>() {
   ♦  @Override
   ♦  public void changed(ObservableValue<? extends String>
      observableValue, String oldValue, String newValue) {
      ◊  If (newValue is null and oldValue is not null) {
          ●  If (the selected Business Name is null) {
```

```
        o  Clear entries for PO Number
        o  Disable PO Number
        o  Clear selected Product
        o  Clear entries for Product
        o  Disable Product
        o  Set poConnectionError to false
    •  } else {
        o  try {
            ♦  Clear the selected Product, its entries, and set it
               to null
            ♦  Set PO Number entries to the new entries for the
               new Business Name value
            ♦  Enable PO Number
            ♦  Set businessConnectionError to false
            ♦  Set poConnectionError to false
        o  } catch (SQLException) {
            ♦  Notify user of error connecting to database and
               tell them they can try again
            ♦  Set businessConnectionError to true
            ♦  Set poConnectionError to false
        o  }
    •  }
◊  } else if (newValue is not null and oldValue is not null)
   {
    •  If (newValue does not equal oldValue) {
        o  If (poRevert is false and selected Product is not
           null) {
            ♦  Ask if user wants to confirm making their change,
               as it will clear Product
            ♦  If (yes) {
                ◊  Clear out the list in Product and set the
                   selected Product to null
                ◊  try {
                    •  Set the new values for Product
                    •  Set poConnectionError to false
                ◊  } catch (SQLException) {
                    •  Notify user of error connecting to database
                       and tell them they can try again
                    •  Set poConnectionError to true
                ◊  }
            ♦  } else {
                ◊  Set poRevert to true
                ◊  Set selected PO Number to oldValue
            ♦  }
        o  } else if (poRevert is false)
            ♦  try {
```

```
                    ◊ Fetch values for Product for the selected PO
                      Number
                    ◊ Set poConnectionError to false
                  ♦ } catch (SQLException) {
                    ◊ Notify user of error connecting to database and
                      tell them they can try again
                    ◊ Set poConnectionError to true
                  ♦ }
              o } else
                  ♦ Set poRevert to false
          ● }
      ◊ } else if (newValue is not null) {
          ● try {
              o Fetch values for Product for the selected PO Number
              o Enable Product
              o Set poConnectionError to false
          ● } catch (SQLException) {
              o Notify user of error connecting to database and tell
                them they can try again
              o Set poConnectionError to true
          ● }
      ◊ }
  ♦ }
o }
o When change is made to CreditReplaceRepair(call new
  ChangeListener<String>() {
  ♦ @Override
  ♦ public void changed(ObservableValue<? extends String>
    observableValue, String oldValue, String newValue) {
    ◊ If (newValue is not null)
      ● If (newValue equals "Credit") {
        o If (Replacement Tracking Number is not empty or blank,
          or Replacement Ship Date is not null, or Ship
          Replacement Repair is true) {
          ♦ Explain to user that by selecting "Credit", they
            will be clearing and disabling the controls under
            Replacement Information because it is not needed,
            and ask if they want to continue
          ♦ If (user wants to continue) {
            ◊ Clear and disable Replacement Tracking Number,
              Replacement Ship Date, and Ship Replacement
              Repair
          ♦ } else
            ◊ Reset to previous value or empty
        o } else {
          ♦ Disable Replacement Tracking Number, Replacement
            Ship Date, and Ship Replacement Repair
```

```
            o  }
          • } else {
            o  Re-enable Replacement Tracking Number, Replacement
               Ship Date, and Ship Replacement Repair
          • }
        ◊ }
    ♦ }
}
```

- cmbCustomerNameOnHidden(event) {
  - cmbCustomerNameOnHidden checks, after the dropdown menu closes, whether the user had experienced a customerConnectionError, and if so, attempts the query again to populate Business Name.
  - If (customerConnectionError)
    - try {
      - Set selected Business Name to null
      - Fetch values for Business Name for the selected Customer Name
      - Enable Business Name control
      - Set customerConnectionError to false
    - } catch (SQLException) {
      - Notify user of error connecting to database and tell them they can try again
      - Set customerConnectionError to true
    - }

```
}
```

- cmbBusinessNameOnHidden(event) {
  - cmbBusinessNameOnHidden checks, after the dropdown menu closes, whether the user had experienced a businessConnectionError, and if so, attempts the query again to populate PO Number.
  - If (businessConnectionError)
    - try {
      - Set selected PO Number to null
      - Fetch values for PO Number for the selected Business Name
      - Enable PO Number
      - Set businessConnectionError to false
    - } catch (SQLException) {
      - Notify user of error connecting to database and tell them they can try again
      - Set businessConnectionError to true
    - }

```
}
```

- cmbPONumberOnHidden(event) {
  - cmbPONumberOnHidden checks, after the dropdown menu closes, whether the user had experienced a poConnectionError, and if so, attempts the query again to populate Product.

- If (poConnectionError)
  - try {
    - Set selected Product to null
    - Fetch values for Product for the selected PO Number
    - Enable Product
    - Set poConnectionError to false
  - } catch (SQLException) {
    - Notify user of error connecting to database and tell them they can try again
    - Set poConnectionError to true
  - }

}

- bntCancelOnAction(event){
  - This will clear and close the form without saving.
  - Change the Confirm buttons to caption "Confirm" and "Cancel"
  - If (pop up confirm select "yes")
    - Clear RMAform and close

}

- btnSaveOnAction(event){
  - This saves the form to the database and creates its RMA Number.
  - If (selected Customer Name is not empty and selected Business Name is not null and selected PO Number is not empty and selected Return Reason Code is not empty and selected Credit Replace Repair value is not empty and selected RMA Status is not empty and selected Product is not null and Return Quantity is > 0) {
    - Get next RMA number
    - Save RMA into database
    - Pop up display "RMA Save Confirm" information
  - } Else
    - Pop up display "Required Fields" error

}

- btnSaveCloseOnAction(event){
  - This save the form to the database and closes the RMAform
  - If (selected Customer Name is not empty and selected Business Name is not null and selected PO Number is not empty and selected Return Reason Code is not empty and selected Credit Replace Repair value is not empty and selected RMA Status is not empty and selected Product is not null and Return Quantity is > 0) {
    - Save RMA into database
    - Pop up display "RMA Save Confirm" information
    - Close form
  - } Else
    - Pop up display "Required Fields" error

}

RMAFormModel

- RMAFormModel() throw exception on error{
  - This constructor creates the form and will initialize all the properties of the RMA new form.
  - Call a getRMADetails method to set the properties

  }
- getRmaDetails(rmaId) {
  - This fetches and displays the RMA details.
  - Get an instance of DBService Class
  - Set all properties and fields of the RMA Form

  }
- List<String> convertHashMapToArrayList(map){
  - This method converts a Map<String, String> into a List<String>.
  - Initialize result as a List of Strings
  - for(each key in map)
    - Add key and its value in "key value" format to result
  - return List

  }

RMADetailsFormController

- loadRMADetails (rmaId) {
  - This method calls the loadRMADetails method in the model, updates the RMA request's age and progress, loads the Details Window with the Customer information, and then signals that initial setup is complete and all future changes in the model should be pushed to the database.

  }

- initialize(url, rb){
  - Initialize ties together the model with the GUI and sets up controls with listeners and event handlers.
  - Create TextFormatter to restrict input to positive integers for Quantity
  - Bind controls to model
  - If (role is engineers) {
    - Restrict control access to Engineering Evaluation
  - } else if (role is analysts)
    - Disable Engineering Evaluation edit button
  - When selected Owner changes(new ChangeListener<String>() {
    - @Override
    - changed(observableValue, oldValue, newValue) {
      - If (we are not doing initialSetup)
        - If (ownerRevert is false) {
          - Try {
            - Update RMA Owner in database
            - update Last Modified details
            - Update RMA Progress
          - } catch (SQLException) {
            - Notify user about error and how to try again

```
                                    ● Set ownerRevert to true
                                    ● Revert Owner to old value
                                ■ }
                            ○ } Else
                                ■ Set ownerRevert to false
                    ■ }
            ○ }
            ○ When RMA Status changes(new ChangeListener<String>() {
                    ■ @Override
                    ■ changed(observableValue, oldValue, newValue) {
                        ● If (we are not doing initialSetup)
                            ○ If (rmaStatusRevert is false) {
                                ■ If (the new value is "Closed") {
                                    ● If (RMA Progress == 1.0) {
                                        ○ Ask user if they want to
                                          continue with closing RMA
                                          because no further changes
                                          can be made after closing
                                          it
                                        ○ If (yes) {
                                            ■ Try {
                                            ■ Update RMA Status to
                                              Closed in database
                                            ■ Disable all controls
                                            ■ } catch
                                              (SQLException) {
                                            ■ Notify user about
                                              error
                                            ■ Set rmaStatusRevert
                                              to true
                                            ■ Revert RMA Status to
                                              old value
                                            ■ }
                                        ○ } Else {
                                            ■ Set rmaStatusRevert
                                              to true
                                            ■ Revert RMA Status to
                                              old value
                                        ○ }
                                    ● } Else {
                                        ○ Notify user that the RMA is
                                          not completely filled out
                                        ○ Set rmaStatusRevert to true
                                        ○ Revert RMA Status to old
                                          value
                                    ● }
                                ■ } Else {
                                    ● try {
                                        ○ Update RMA Status in
                                          database to new value
                                        ○ Update Last Modified
                                          details
```

```
                                    ○ Update RMA Progress
                                ● } catch (SQLException) {
                                    ○ Notify user about error
                                    ○ Set rmaStatusRevert to true
                                    ○ Revert RMA Status to old
                                      value
                                ● }
                            ■ }
                        ○ } Else
                            ■ Set rmaStatusRevert to false
                    ● }
                ■ }
        ○ }
        ○ When Credit Replace Repair changes(new
          ChangeListener<String>() {
            ■ @Override
            ■ changed(observableValue, oldValue, newValue) {
                ● If (we are not doing initialSetup) {
                    ○ If (creditReplaceRepairRevert is false) {
                        ■ If (new value is "Credit") {
                            ● If (Replacement Tracking Number
                              is not empty or blank, or
                              Replacement Ship Date is not
                              null, or Ship Replacement Repair
                              is true) {
                                ○ Ask user if they wish to
                                  continue with the change
                                  because it will clear out
                                  Replacement Tracking
                                  Number, Replacement Ship
                                  Date, and Ship Replacement
                                  Repair due to credits not
                                  needing anything shipped
                                  back
                                ○ If (yes) {
                                    ■ Try {
                                    ■ Update Credit Replace
                                      Repair in database
                                    ■ Set mentioned
                                      controls to empty,
                                      null, or false, as
                                      appropriate, and
                                      update their values
                                      in the database to
                                      reflect the new
                                      values
                                    ■ Disable the mentioned
                                      controls
                                    ■ Collapse the
                                      Replacement Detail
                                      section and disable
```

```
                                the TitledPane so the
                                user cannot expand it
                            ■ Update Last Modified
                                details
                            ■ Update RMA Progress
                            ■ } catch
                                (SQLException) {
                            ■ Notify user about the
                                error
                            ■ Set
                                creditReplaceRepairRe
                                vert to true
                            ■ Set Credit Replace
                                Repair to old value
                            ■ }
                        ○ } Else {
                            ■ Set
                                creditReplaceRepairRe
                                vert to true
                            ■ Set Credit Replace
                                Repair to old value
                        ○ }
                • } Else {
                    ○ Try {
                        ■ Update Credit Replace
                            Repair in database
                        ■ Disable the mentioned
                            controls
                        ■ Collapse the
                            Replacement Detail
                            section and disable
                            the TitledPane so the
                            user cannot expand it
                        ■ Update Last Modified
                            details
                        ■ Update RMA Progress
                    ○ } catch (SQLException) {
                        ■ Notify user about the
                            error
                        ■ Set
                            creditReplaceRepairRe
                            vert to true
                        ■ Set Credit Replace
                            Repair to old value
                    ○ }
                • }
            ■ } Else {
                • Try {
                    ○ Update Credit Replace
                        Repair in database
                    ○ Re-enable Replacement
                        Tracking Number,
```

```
                                        Replacement Ship Date, Ship
                                        Replacement Repair, and
                                        Replacement Detail
                                        TitledPane, and expand
                                        TitledPane
                                    ○ Update Last Modified
                                        Details
                                    ○ Update RMA Progress
                                ● } catch (SQLException) {
                                    ○ Notify user about the error
                                    ○ Set
                                        creditReplaceRepairRevert
                                        to true
                                    ○ Set Credit Replace Repair
                                        to old value
                                ● }
                            ■ }
                        ○ } Else
                            ■ Set creditReplaceRepairRevert = false
                ● } Else {
                    ○ Disable Replacement Tracking Number,
                        Replacement Ship Date, and Ship Replacement
                        Repair
                    ○ Collapse the Replacement Detail section and
                        disable the TitledPane so the user cannot
                        expand it
                ● }
            ■ }
    ○ }
    ○ When Return Reason Code changes(new
        ChangeListener<String>() {
            ■ @Override
            ■ changed(observableValue, oldValue, newValue) {
                ● If (we are not doing initialSetup)
                    ○ If (returnReasonCodeRevert is false) {
                        ■ Try {
                            ● Update Return Reason Code in
                                database
                            ● Update Last Modified details
                            ● Update RMA Progress
                        ■ } catch (SQLException) {
                            ● Notify user about error
                            ● Set returnReasonCodeRevert to
                                true
                            ● Set Return Reason Code to old
                                value
                        ■ }
                    ○ } Else
                        ■ Set returnReasonCodeRevert to false
            ■ }
    ○ }
```

```
    ○ When Product changes(new
      ChangeListener<PurchaseOrderProduct>() {
        ■ @Override
        ■ changed(observableValue, oldValue, newValue) {
            ● If (we are not doing initialSetup)
                ○ If (productRevert is false) {
                    ■ Try {
                        ● Update assigned RMA Product in
                          database
                        ● Update Last Modified details
                        ● Update RMA Progress
                    ■ } catch (SQLException) {
                        ● Notify user about error
                        ● Set productRevert to true
                        ● Set Product to old value
                    ■ }
                ○ } Else
                    ■ Set productRevert to false
        ■ }
    ○ }
    ○ When Disposition changes(new ChangeListener<String>() {
        ■ @Override
        ■ changed(observableValue, oldValue, newValue) {
            ● If (we are not doing initialSetup)
                ○ If (dispositionRevert is false) {
                    ■ Try {
                        ● Update RMA Disposition in the
                          database
                        ● Update Last Modified details
                        ● Update RMA Progress
                    ■ } catch (SQLException) {
                        ● Notify user about the error
                        ● Set dispositionRevert to true
                        ● Set Disposition to old value
                    ■ }
                ○ } Else
                    ■ Set dispositionRevert to false
        ■ }
    ○ }
    ○ When RMA Progress changes(new ChangeListener<Number>() {
        ■ @Override
        ■ changed(observableValue, oldValue, newValue) {
            ● If (new value is 1.0 and RMA Status is not
              "Closed") {
                ○ Notify user that they can close the RMA
            ● }
        ■ }
    ○ When Ship Replacement Repair changes(new
      ChangeListener<Boolean>() {
        ■ @Override
        ■ changed(observableValue, oldValue, newValue) {
            ● If (we are not doing initialSetup)
```

```
                    ○ If (cbReplacementRepairShip1Revert is
                      false)
                        ■ Try {
                            ● Update Ship Replacement Repair
                              in database
                            ● Update Last Modified details
                            ● Update RMA Progress
                        ■ } catch (SQLException) {
                            ● Notify user about the error
                            ● Set
                              cbReplacementRepairShip1Revert
                              to true
                            ● Set Ship Replacement Repair to
                              old value
                        ■ }
                    ○ Else
                        ■ Set cbReplacementRepairShip1Revert to
                          false
        ■ }
○ }
○ When Replacement Repair Ship Date changes(new
  ChangeListener<Number>() {
        ■ @Override
        ■ changed(observableValue, oldValue, newValue) {
            ● If (we are not doing initialSetup)
                ○ If (dpReplacementRepairDate1Revert is
                  false)
                    ■ Try {
                        ● Update RMA Replacement Ship Date
                          in database
                        ● Update Last Modified details
                        ● Update RMA Progress
                    ■ } catch (SQLException) {
                        ● Notify user of error
                        ● Set
                          dpReplacementRepairDate1Revert
                          to true
                        ● Set Replacement Repair Ship Date
                          to old value
                    ■ }
                ○ Else
                    ■ Set dbReplacementRepairDate1Revert to
                      true
        ■ }
○ }
○ When the text in the Replacement Repair Ship Date text
  editor changes(new ChangeListener<String>() {
        ■ @Override
        ■ changed(observableValue, oldValue, newValue) {
            ● If (we are not doing initialSetup)
                ○ If (the new value is empty)
```

```
                    ■ Set Replacement Repair Ship Date to
                      null
        ■ }
    o }
    o When the Information TitledPane expanded status changes(new
      ChangeListener<Boolean>() {
        ■ @Override
        ■ changed(observableValue, oldValue, newValue) {
            ● If new value is false, set max height to 0.0,
              else set it to tpInformationPrefHeight
        ■ }
    o }
    o When the Product Information TitledPane expanded status
      changes(new ChangeListener<Boolean>() {
        ■ @Override
        ■ changed(observableValue, oldValue, newValue) {
            ● If new value is false, set max height to 0.0,
              else set it to tpProductPrefHeight
        ■ }
    o }
    o When the Product Evaluation TitledPane expanded status
      changes(new ChangeListener<Boolean>() {
        ■ @Override
        ■ changed(observableValue, oldValue, newValue) {
            ● If new value is false, set max height to 0.0,
              else set it to tpEvaluationPrefHeight
        ■ }
    o }
    o When the Product Disposition TitledPane expanded status
      changes(new ChangeListener<Boolean>() {
        ■ @Override
        ■ changed(observableValue, oldValue, newValue) {
            ● If new value is false, set max height to 0.0,
              else set it to tpDispositionPrefHeight
        ■ }
    o }
    o When Replacement Detail TitledPane expanded status
      changes(new ChangeListener<Boolean>() {
        ■ @Override
        ■ changed(observableValue, oldValue, newValue) {
            ● If new value is false, set max height to 0, else
              set it to tpReplacementRepairPrefHeight
        ■ }
    o }
    o Set values for tpInformationPrefHeight,
      tpProductPrefHeight, tpEvaluationPrefHeight,
      tpDispositionPrefHeight, and tpReplacementRepairPrefHeight
    o When Owner CheckBox is showing(new EventHandler<Event>() {
        ■ @Override
        ■ handle(event) {
            ● If (user's role is "engineers")
                o Hide CheckBox menu
```

```
            }
        }
        When RMA Status CheckBox is showing(new
        EventHandler<Event>() {
            @Override
            handle(event) {
                If (user's role is "engineers")
                    Hide CheckBox menu
            }
        }
        When Credit Replace Repair CheckBox is showing(new
        EventHandler<Event>() {
            @Override
            handle(event) {
                If (user's role is "engineers")
                    Hide CheckBox menu
            }
        }
        When Return Reason Code CheckBox is showing(new
        EventHandler<Event>() {
            @Override
            handle(event) {
                If (user's role is "engineers")
                    Hide CheckBox menu
            }
        }
        When Product CheckBox is showing(new EventHandler<Event>()
        {
            @Override
            handle(event) {
                If (user's role is "engineers")
                    Hide CheckBox menu
            }
        }
        When Disposition CheckBox is showing(new
        EventHandler<Event>() {
            @Override
            handle(event) {
                If (user's role is "engineers")
                    Hide CheckBox menu
            }
        }
        When Replacement Repair Ship Date DatePicker is showing(new
        EventHandler<Event>() {
            @Override
            handle(event) {
                If (user's role is "engineers")
                    Hide DatePicker menu
            }
        }
}
```

- handleHplCustomerNameDetails1OnAction(){
  - This method displays the Customer info and PO Numbers in the Details Window.

  }

- handleHplPoNumberOnAction(event){
  - This method displays info about the Purchase Order Products for the selected PO Number in the Details Window.

  }

- btnSpecialInstructionEdit1OnAction(event) {
  - This method handles making the Special Instructions/Additional Info TextArea modifiable and saving the changes to the database.
  - If (the btnSpecialInstructionEdit1 text says "Edit") {
    - If (ask if the user wants to modify the field using editConfirmationMessage) {
      - Set field to be editable and disable all other controls that modify or are modifiable
    - }
  - } Else {
    - If (ask if the user wants to save the field using editConfirmationMessage) {
      - Try {
        - Update RMA Additional Info in database
        - Update Last Modified details
        - Update RMA Progress
        - Set field to not be editable and re-enable all other controls (Engineering Evaluation depending on whether user is part of "admins", and the Replacement Detail section depending on if RMA Status is "Credit")
      - } catch (SQLException) {
        - Notify user of error
      - }
    - }
  - }

  }

- btnReturnLabelTrackingEdit1OnAction(event) {
  - This method handles making the Return Label Tracking # TextField modifiable and saving the changes to the database.
  - If (the btnReturnLabelTrackingEdit1 text says "Edit") {
    - If (ask if the user wants to modify the field using editConfirmationMessage) {
      - Set field to be editable and disable all other controls that modify or are modifiable
    - }
  - } Else {

            - If (ask if the user wants to save the field using editConfirmationMessage) {
                - Try {
                    - Update RMA Return Label Tracking # in database
                    - Update Last Modified details
                    - Update RMA Progress
                    - Set field to not be editable and re-enable all other controls (Engineering Evaluation depending on whether user is part of "admins", and the Replacement Detail section depending on if RMA Status is "Credit")
                - } catch (SQLException) {
                    - Notify user of error
                - }
            - }
        - }
    }

- btnQuantityEdit1OnAction(event) {
    - This method handles making the Return Quantity TextField modifiable and saving the changes to the database.
    - If (the btnQuantityEdit1 text says "Edit") {
        - If (ask if the user wants to modify the field using editConfirmationMessage) {
            - Set field to be editable and disable all other controls that modify or are modifiable
        - }
    - } Else {
        - Try {
            - Update RMA Return Quantity in database
            - Update Last Modified details
            - Update RMA Progress
            - Set field to not be editable and re-enable all other controls (Engineering Evaluation depending on whether user is part of "admins", and the Replacement Detail section depending on if RMA Status is "Credit")
        - } catch (SQLException) {
            - Notify user of error
        - }
    - }
  }
- btnInitialEvaluationEdit1OnAction(event) {
    - This method handles making the Initial Evaluation TextArea modifiable and saving the changes to the database.
    - If (the btnInitialEvaluationEdit1 text says "Edit") {
        - If (ask if the user wants to modify the field using editConfirmationMessage) {
            - Set field to be editable and disable all other controls that modify or are modifiable

        - }
    - } Else {
        - If (ask if the user wants to save the field using editConfirmationMessage) {
            - Try {
                - Update RMA Initial Evaluation in database
                - Update Last Modified details
                - Update RMA Progress
                - Set field to not be editable and re-enable all other controls (Engineering Evaluation depending on whether user is part of "admins", and the Replacement Detail section depending on if RMA Status is "Credit")
            - } catch (SQLException) {
                - Notify user of error
            - }
        - }
    - }
}

- btnEngineeringEvaluationEdit1OnAction(event) {
    - This method handles making the Engineering Evaluation TextArea modifiable and saving the changes to the database.
    - If (the btnEngineeringEvaluationEdit1 text says "Edit") {
        - If (ask if the user wants to modify the field using editConfirmationMessage) {
            - Set field to be editable and disable all other controls that modify or are modifiable
        - }
    - } Else {
        - If (ask if the user wants to save the field using editConfirmationMessage) {
            - Try {
                - Update RMA Engineering Evaluation in database
                - Update Last Modified details
                - Update RMA Progress
                - Set field to not be editable and re-enable all other controls (the Replacement Detail section depending on if RMA Status is "Credit")
            - } catch (SQLException) {
                - Notify user of error
            - }
        - }
    - }
}

- btnDispositionNotesEdit1OnAction(event) {
  - This method handles making the Disposition Notes TextArea modifiable and saving the changes to the database.
  - If (the btnDispositionNotesEdit1 text says "Edit") {
    - If (ask if the user wants to modify the field using editConfirmationMessage) {
      - Set field to be editable and disable all other controls that modify or are modifiable
    - }
  - } Else {
    - If (ask if the user wants to save the field using editConfirmationMessage) {
      - Try {
        - Update RMA Disposition Notes in database
        - Update Last Modified details
        - Update RMA Progress
        - Set field to not be editable and re-enable all other controls (Engineering Evaluation depending on whether user is part of "admins", and the Replacement Detail section depending on if RMA Status is "Credit")
      - } catch (SQLException) {
        - Notify user of error
      - }
    - }
  - }

}

- btnReplacementRepairTrackingEdit1OnAction(event) {
  - This method handles making the Replacement Repair Tracking # TextField modifiable and saving the changes to the database.
  - If (the btnReplacementRepairTrackingEdit1 text says "Edit") {
    - If (ask if the user wants to modify the field using editConfirmationMessage) {
      - Set field to be editable and disable all other controls that modify or are modifiable
    - }
  - } Else {
    - If (ask if the user wants to save the field using editConfirmationMessage) {
      - Try {
        - Update RMA Replacement Repair Tracking # in database
        - Update Last Modified details
        - Update RMA Progress
        - Set field to not be editable and re-enable all other controls (Engineering Evaluation depending on whether user is part of "admins")

- } catch (SQLException) {
  - Notify user of error
- }
  - }
- }
}

- boolean editConfirmationMessage(btnName, name, editSave){
  - This method displays a custom confirmation message for handling the Buttons that allow the user to modify and save modifications to TextFields and TextAreas.
  - Declare msg String to hold the result
  - if (btnName text is "Save")
    - Set msg to "Would you like to save changes to the "
  - Else
    - Set msg to "Would you like to edit the "
  - Define a new Alert instance using the msg variable and name parameter
  - Show prompt to user
  - If (yes) {
    - Change the btnName Button Text to editSave parameter
    - Return true
  - } Else
    - Return false

}

RMADetailsFormModel

- RMADetailsFormModel() {
  - Constructor that initializes a new RMADetailsFormModel.
  - Call super to populate the fields inherited from RMAFormModel
  - Initialize fields unique to RMADetailsFormModel

  }

- getRMADetails(rmaId) {
  - Loads RMADetailsFormModel with the database details for the given RMA ID.
  - Fetch instance of DBService
  - Populate details and fetch possible selections based on details for the passed-in RMA ID

  }

- updateRMAProgress() {
  - Updates the progress for the RMA based on whether the RMA's Credit Replace Repair value is Credit because credits do not require a return shipment.
  - If (RMA Status equals Credit) {
    - create double variable called progress to track progress
    - If (selected Customer Name is not empty or blank and the selected Business Name is not null and the selected PO Number is not empty or blank and the selected Owner is not empty or

```
        blank and the selected RMA Status is not empty or blank and
        the selected Credit Replace Repair option is not empty or
        blank and the selected Return Reason Code is not empty or
        blank)
        ◊ Add 20.0 to progress
    ♦ If (selected Product is not null and Return Label Tracker is
      not empty or null and the Return Quantity is greater than 0)
        ◊ Add 20.0 to progress
    ♦ If (Initial Evaluation is not empty or blank and Engineering
      Evaluation is not empty or blank)
        ◊ Add 20.0 to progress
    ♦ If (selected Disposition is not empty or blank and Disposition
      Notes is not empty or blank)
        ◊ Add 20.0 to progress
    ♦ if (Replacement Tracking Number is not empty or blank and
      Replacement Ship Date is not null and Ship Replacement Repair
      is true)
        ◊ Add 20.0 to progress
    ♦ Set RMA progress to progress / 100.0
o } else {
    ♦ create double variable called progress to track progress
    ♦ If (selected Customer Name is not empty or blank and the
      selected Business Name is not null and the selected PO Number
      is not empty or blank and the selected Owner is not empty or
      blank and the selected RMA Status is not empty or blank and
      the selected Credit Replace Repair option is not empty or
      blank and the selected Return Reason Code is not empty or
      blank)
        ◊ Add 25.0 to progress
    ♦ If (selected Product is not null and Return Label Tracker is
      not empty or null and the Return Quantity is greater than 0)
        ◊ Add 25.0 to progress
    ♦ If (Initial Evaluation is not empty or blank and Engineering
      Evaluation is not empty or blank)
        ◊ Add 25.0 to progress
    ♦ If (selected Disposition is not empty or blank and Disposition
      Notes is not empty or blank)
        ◊ Add 25.0 to progress
    ♦ Set RMA progress to progress / 100.0
o }
}
```

3. Decision Tables

Login

	1	2	3	4	5	6	7	8	9	10	11
Condition											
Username	T	T	F	T	F	F			T	F	
Password	T	T	T	F	F	F	T	F			
Instance Selected	T	T	T	T	T	F	T	T	T	T	T
Connection to Server Established	T	F	x	x	x	x	x	x	x	x	x
Action											
Login Succcessful	Execute										
Home Screen Displays	Execute										
Error Show Invalid Credential			Execute	Execute	Execute	Execute					
Error Show Missing Credential Input							Execute	Execute	Execute	Execute	Execute
Error Show Connection to Server Error		Execute									

Home Screen

	Refresh Button Selected																															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Condition																																
Search Field Contains Value	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
Non-Critical RMA Exist	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F
Critical RMA Exist	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F
New RMA Saved	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F
RMA Selected	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F
Action																																
New RMA Form Display																																
RMA Detail Screen Displays																																
List Refresh	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute
Update Indicator	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute
Clear RMA Selected	Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute	
Clear Search Field	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute																
Delete Confirmation Pop Up Displays																																
Critical RMA Exist Pop Up Displays																																

	New RMA Button Selected																													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Condition																																
Search Field Contains Value	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
Non-Critical RMA Exist	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F
Critical RMA Exist	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F
New RMA Saved	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F
RMA Selected	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F
Action																																
New RMA Form Display	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute
RMA Detail Screen Displays																																
List Refresh	Execute	Execute			Execute	Execute			Execute	Execute			Execute	Execute			Execute	Execute			Execute	Execute			Execute	Execute			Execute	Execute		
Update Indicator	Execute	Execute			Execute	Execute			Execute	Execute			Execute	Execute			Execute	Execute			Execute	Execute			Execute	Execute			Execute	Execute		
Clear RMA Selected	Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute	
Clear Search Field	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute																
Delete Confirmation Pop Up Displays																																
Critical RMA Exist Pop Up Displays																																

	Delete Button Selected																															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Condition																																
Search Field Contains Value	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
Non-Critical RMA Exist	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F
Critical RMA Exist	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F
New RMA Saved	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F
RMA Selected	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F
Action																																
New RMA Form Display																																
RMA Detail Screen Displays																																
List Refresh																																
Update Indicator																																
Clear RMA Selected																																
Clear Search Field	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute																
Delete Confirmation Pop Up Displays	Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute		Execute	
Critical RMA Exist Pop Up Displays																																

	Critical RMA Selected																													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Condition																																
Search Field Contains Value	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
Non-Critical RMA Exist	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F
Critical RMA Exist	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F
New RMA Saved	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F
RMA Selected	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F
Action																																
New RMA Form Display																																
RMA Detail Screen Displays	Execute	Execute	Execute	Execute	x	x	x	x	Execute	Execute	Execute	Execute	x	x	x	x	Execute	Execute	Execute	Execute	x	x	x	x	Execute	Execute	Execute	Execute	x	x	x	x
List Refresh																																
Update Indicator																																
Clear RMA Selected																																
Clear Search Field																																
Delete Confirmation Pop Up Displays																																
Critical RMA Exist Pop Up Displays																																

	Non-Critical RMA Selected																															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Condition																																
Search Field Contains Value	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
Non-Critical RMA Exist	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F
Critical RMA Exist	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F
New RMA Saved	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F
RMA Selected	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F
Action																																
New RMA Form Display																																
RMA Detail Screen Displays					Execute	Execute	Execute	Execute	x	x	x	x	x	x	x	x					Execute	Execute	Execute	Execute	x	x	x	x	x	x	x	x
List Refresh																																
Update Indicator																																
Clear RMA Selected																																
Clear Search Field																																
Delete Confirmation Pop Up Displays																																
Critical RMA Exist Pop Up Displays	Execute	Execute	Execute	Execute					Execute	Execute	Execute	Execute					Execute	Execute	Execute	Execute					Execute	Execute	Execute	Execute				

	Delete Confirmation Pop Up							
	1	2	3	4	5	6	7	8
Condition								
Confirm Selected	T	T	x	x	x	x	x	x
Cancel Selected	x	x	T	T	x	x	x	x
Close Pop Up Selected	x	x	x	x	T	T	x	x
New RMA Saved	T	F	T	F	T	F	T	F
Action								
List Refresh	Execute	Execute	Execute		Execute			
Update Indicator	Execute	Execute	Execute		Execute			
Delete Confirmation Pop Up Close	Execute	Execute	Execute	Execute	Execute	Execute		
Clear RMA Selected	Execute	Execute						

	Search Field Selected							
	1	2	3	4	5	6	7	8
Condition								
Valid RMA Number Entered	T	T	T	T	F	F	F	F
New RMA Saved	T	T	F	F	T	T	F	F
RMA Selected	T	F	T	F	T	F	T	F
Action								
New RMA Form Display								
RMA Detail Screen Displays								
List Refresh	Execute	Execute	Execute				Execute	Execute
Update Indicator	Execute	Execute	Execute				Execute	Execute
Delete Confirmation Pop Up Displays								
Clear RMA Selected	Execute	Execute	Execute	Execute				
Critical RMA Exist Pop Up Displays								

	Critical RMA Exist Pop Up					
	1	2	3	4	5	6
Condition						
OK Selected	T	T	x	x	x	x
Close Pop Up Selected	x	x	T	T	x	x
New RMA Saved	T	F	T	F	T	F
Action						
List Refresh	Execute		Execute			
Update Indicator	Execute		Execute			
Critical RMA Exist Pop Up Close	Execute	Execute	Execute	Execute		
Clear RMA Selected						
Clear Search Field						

NewRMA

	Save Button Selected							
	1	2	3	4	5	6	7	8
Condition								
All Required Fields Completed	T	T	T	T	F	F	F	F
Non Required Fields Completed	T	T	F	F	T	T	F	F
Quantity Value > 0	T	F	T	F	T	F	T	F
Action								
New RMA Form Close								
RMA Saved into Database Pop Displays	Execute		Execute					
RMA Saved into Database	Execute		Execute					
List Refresh	Execute		Execute					
Update Indicator	Execute		Execute					
Clear RMA Selected								
Clear Search Field								
Error Pop up Required Displays		Execute		Execute	Execute	Execute	Execute	Execute
Close and Clear Form Warning Displays								
Home Screen Displays								

	Save and Close Button Selected							
	1	2	3	4	5	6	7	8
Condition								
All Required Fields Completed	T	T	T	T	F	F	F	F
Non Required Fields Completed	T	T	F	F	T	T	F	F
Quantity Value > 0	T	F	T	F	T	F	T	F
Action								
New RMA Form Close	Execute		Execute					
RMA Saved into Database Pop Displays	Execute		Execute					
RMA Saved into Database	Execute		Execute					
List Refresh	Execute		Execute					
Update Indicator	Execute		Execute					
Clear RMA Selected	Execute		Execute					
Clear Search Field	Execute		Execute					
Error Pop up Required Displays		Execute		Execute	Execute	Execute	Execute	Execute
Close and Clear Form Warning Displays								
Home Screen Displays	Execute		Execute					

	Cancel Button Selected							
	1	2	3	4	5	6	7	8
Condition								
All Required Fields Completed	T	T	T	T	F	F	F	F
Non Required Fields Completed	T	T	F	F	T	T	F	F
Quantity Value > 0	T	F	T	F	T	F	T	F
Action								
New RMA Form Close	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute
RMA Saved into Database Pop Displays								
RMA Saved into Database								
List Refresh								
Update Indicator								
Clear RMA Selected								
Clear Search Field								
Error Pop up Required Displays								
Close and Clear Form Warning Displays	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute
Home Screen Displays	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute

	Close Button Selected							
	1	2	3	4	5	6	7	8
Condition								
All Required Fields Completed	T	T	T	T	F	F	F	F
Non Required Fields Completed	T	T	F	F	T	T	F	F
Quantity Value > 0	T	F	T	F	T	F	T	F
Action								
New RMA Form Close								
RMA Saved into Database Pop Displays								
RMA Saved into Database								
List Refresh								
Update Indicator								
Clear RMA Selected								
Clear Search Field								
Error Pop up Required Displays								
Close and Clear Form Warning Displays	Execute	Execute	Execute	Execute	Execute	Execute	Execute	Execute

	Error Required Displays		
	1	2	3
Condition			
OK Selected	T	x	x
Close Pop Up Selected	x	T	x
Action			
Error Required Pop Up Close	Execute	Execute	
Highlight Missing Required Field	Execute	Execute	

	Saved Confirmation Displays			
	1	2	3	4
Condition				
OK Selected	T	T	x	x
Close Pop Up Selected	x	x	T	T
Save Button Selected	x	T	x	T
Save and Closed Selected	T	x	T	x
Action				
Save Pop Up Close	Execute	Execute	Execute	
Home Screen Displays	Execute		Execute	

	Close and Clear Warning Displays			
	1	2	3	4
Condition				
Confirm Selected	T	x	x	x
Cancel Selected	x	T	x	x
Close Pop Up Selected	x	X	T	x
Action				
Error Required Pop Up Close	Execute	Execute	Execute	
RMA Form Reset	Execute			
Home Screen Displays	Execute			

RMADetail

	Saved button Is Selected															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Condition																
Required Fields are Complete	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F
Product Disposition Section Complete	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F
Confirm Save Selected	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F
Progress bar is at fullfilled state	T	T	T	T	T	T	T	T	x	x	x	x	x	x	x	x
Progress bar is > 0% complete	x	x	x	x	x	x	x	x	T	T	T	T	T	T	T	T
Action																
Progress bar updates increments 20%									Execute							
Progress bar updates decrements 20%											Execute					
Progress bar updates to Fulfilled																
RMA updated and Saved	Execute								Execute		Execute					
Last Modified By Update to Current User	Execute								Execute		Execute					
Saved Button changed to Edit button	Execute								Execute		Execute					
Save Pop-up Displays	Execute								Execute		Execute					
Error Pop Up Displays		Execute	Execute	Execute						Execute		Execute				
Confirm Save Pop up Hides					Execute	Execute	Execute	Execute					Execute	Execute	Execute	Execute

	Saved button Is Selected															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Condition																
Required Fields are Complete	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F
Replacement Detail Section Complete	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F
Confirm Save Selected	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F
Progress bar is at fullfilled state	T	T	T	T	T	T	T	T	x	x	x	x	x	x	x	x
Progress bar is > 0% complete	x	x	x	x	x	x	x	x	T	T	T	T	T	T	T	T
Action																
Progress bar updates increments 20%																
Progress bar updates decrements 20%																
Progress bar updates to Fulfilled									Execute							
RMA updated and Saved	Execute								Execute		Execute					
Last Modified By Update to Current User	Execute								Execute		Execute					
Saved Button changed to Edit button	Execute								Execute		Execute					
Save Pop-up Displays	Execute								Execute		Execute					
Error Pop Up Displays		Execute	Execute	Execute						Execute		Execute				
Confirm Save Pop up Hides					Execute	Execute	Execute	Execute					Execute	Execute	Execute	Execute

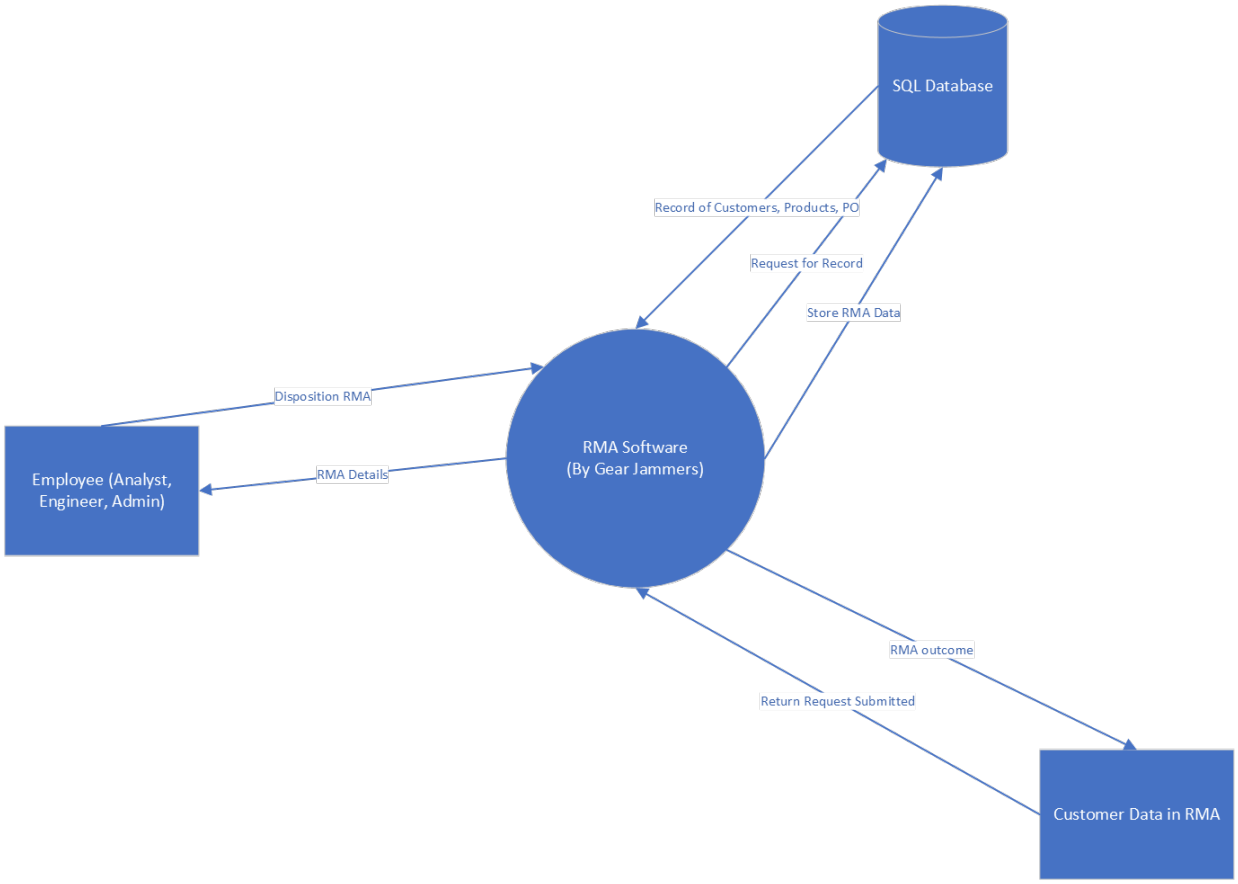
	Edit Button Selected			
	1	2	3	4
Condition				
Every Section Besides Replacement Section Complete	T	F	T	F
Located in the Replacement Detail Window	T	T	F	F
Action				
Edit button changes caption to "Save" button	Execute		Execute	Execute
Buttons and fields are Disabled	Execute		Execute	Execute
Field is Editable	Execute		Execute	Execute

	Hyperlink Selected	
	1	2
Condition		
Information available in Database	T	F
Action		
Field Information populates in the detail window	Execute	

	Home Selected	
	1	2
Condition		
RMA is Complete	T	F
Action		
Return to the Home Screen	Execute	Execute

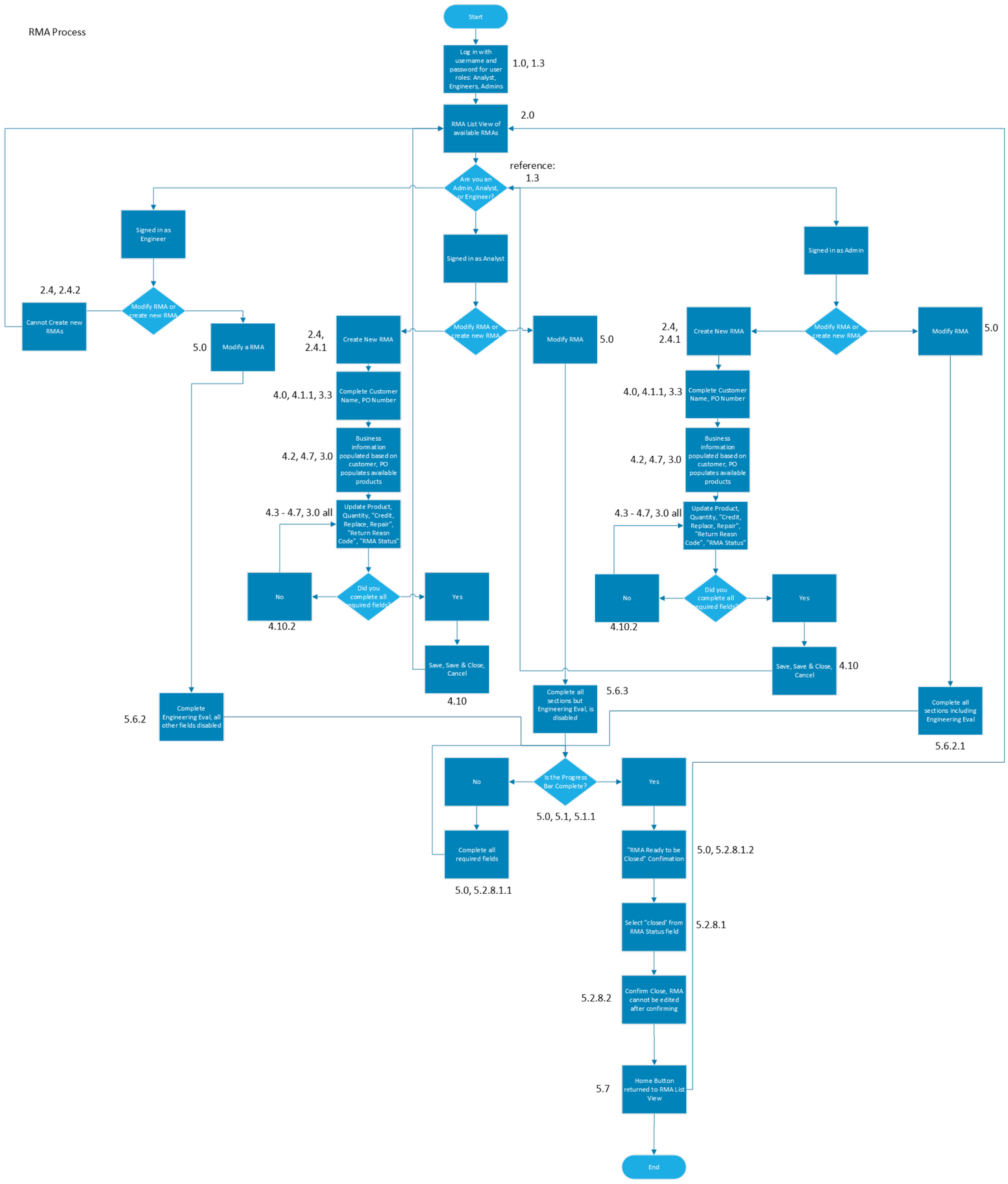
	Tilted Panel Selected		Tilted Panel Selected	
	1	2	3	4
Condition				
Saved is available	T	F	T	F
Section Selected is Expanded	T	T	F	F
Action				
Section Expands			Execute	Execute
Section Contracts	Execute	Execute		

4. Control Diagrams



Employee
RMA Interface
Database
Login
Login
[If Credential Valid]
[Else]
Verify Login
Credential Valid
Access Granted
Credential Invalid
Renter Login
Populate list of all available RMA
loop
[While RMA = Invalid or New RMA = False]
RMA
[If New Return]
[Else]
Create New RMA
New RMA Form Displays
Auto Assign Create By
Default RMA Owner
Enter RMA Number Search field
Search RMA Number
RMA
[If RMA found]
[Else]
RMA Found
RMA Populate in List
RMA Not found
RMA List Empty
RMA Number Invalid
RMA Form
[If New RMA button selected]
Require Fields
[If Following Fields are complete]
Select Business Name Dropdown
Business Name
Business Name List
Cascade List of Customer Name
Search Customer Name
Retrieve Customer Name
Customer Name
Cascade List of Business Name
Customer Shipping Address Populates
Select Select RMA Status
Retrieve RMA Status
RMA Status List
Select Product Dropdown
Retrieves Product
Product List
Enter Return Quantity
Select Save button
Last Modified by Login Info
Last Modified Date = Today
Save to RMA List
Select Save And Close button
Last Modified by Login Info
Last Modified Date = Today
Save to RMA List
Close RMA Form
[Else]
Indicate all Required Fields
Select Reason code Dropdown
Retrieves Reason Codes List
Select Credit Repair Request
Enter P.O. Number
Enter Additional Info
Enter Return Label Tracking #
Initial Evaluation
Engineering Evaluation
Select Disposition Dropdown
Retrieves Disposition List
Disposition List
Enter Disposition notes
Enter Replacement/Repair Tracking #
Select Ship Date
Calendar Populates
Select Cancel button
Close RMA Form
Return to Home Screen
Select RMA from list
Retrieve Selected RMA
RMA
Details
[If RMA From List is Selected]
RMA Detail Screen Displays
Details Window Defaults to Customer
Select Closed Tilted Pane Section
Section Detail Display
Select Open Tilted Pane Section
Section Detail Hides
Edit button Selected
Edit Button Changes to Save
Update Made to field
Selected Saved Button
Last Modified by Login Info
Last Modified Date = Today
Save Update to RMA
Select Customer to view Details
Retrieve Customer Info
Customer Info
Display Customer Info on Details Window
Select Business to view Details
Retrieve Business Info
Business Info
Display Business Info on Details Window
Select Owner to view Details
Retrieve Employee Info
Employee Info
Display Employee Info on Details Window
Select Product to view Details
Retrieve Product Info
Product Info
Display Product Info on Details Window
Attach Files (Optional)
Last Modified by Login Info
Last Modified Date = Today
Update RMA
Select Refresh button
Retrieves Updated RMA List
RMA list
Refreshed RMA List Table
Refresh Indicator Update
Delete RMA
[CheckBox= True]
Select Delete button
Delete Confirmation Pop Up displays
alt
[If Yes is selected]
[Else]
Remove RMA from List
Delete Confirmation Pop Up closes
Update RMA List
Update RMA List Table
Delete Confirmation Pop Up closes
Fulfillment Complete
RMA ready for fullfillment

RMA Process


Start
Log in with username and password for user roles: Analyst, Engineers, Admins
1.0, 1.3
RMA List View of available RMAs
2.0
reference: 1.3
Are you an Admin, Analyst, or Engineer?
Signed in as Engineer
Signed in as Analyst
Signed in as Admin
2.4, 2.4.2
Modify RMA or create new RMA
Cannot Create new RMAs
5.0
Modify a RMA
2.4, 2.4.1
Create New RMA
Modify RMA
5.0
4.0, 4.1.1, 3.3
Complete Customer Name, PO Number
4.2, 4.7, 3.0
Business information populated based on customer, PO populates available products
4.3 - 4.7, 3.0 all
Update Product, Quantity, "Credit, Replace, Repair", "Return Reasn Code", "RMA Status"
Did you complete all required fields?
No
4.10.2
Yes
Save, Save & Close, Cancel
4.10
5.6.2
Complete Engineering Eval, all other fields disabled
5.6.3
Complete all sections but Engineering Eval, is disabled
Complete all sections including Engineering Eval
5.6.2.1
Is the Progress Bar Complete?
5.0, 5.1, 5.1.1
Complete all required fields
5.0, 5.2.8.1.1
"RMA Ready to be Closed" Confimation
5.0, 5.2.8.1.2
Select "closed" from RMA Status field
5.2.8.1
5.2.8.2
Confirm Close, RMA cannot be edited after confirming
5.7
Home Button returned to RMA List View
End

Source Code Documentation

Main.java

```
1.  package RMA;
2.
3.  public class Main {
4.      public static void main(String[] args){RMA.main(args);}
5.  }
6.
```

RMA.java

```
1.  package RMA;
2.
3.  import javafx.application.Application;
4.  import javafx.fxml.FXMLLoader;
5.  import javafx.scene.Scene;
6.  import javafx.stage.Stage;
7.
8.  import java.io.IOException;
9.
10. /** The RMA class starts the JavaFX runtime and launches the Login screen.
11.  */
12. public class RMA extends Application {
13.
14.     /** start opens the Login screen.
15.      * @param stage The initial {@link Stage} passed from the JavaFX runtime.
16.      * @throws IOException If there is an issue finding or loading the Login FXML file or
    Controller.
17.      */
18.     @Override
19.     public void start(Stage stage) throws IOException {
20.         FXMLLoader loader = new FXMLLoader();
21.         loader.setLocation(getClass().getResource("Login_V2.fxml"));
22.         Scene scene = new Scene(loader.load());
23.         stage.setScene(scene);
24.         stage.setTitle("RMA Login");
25.         stage.show();
26.     }
27.
28.     /** main starts the JavaFX runtime.
29.      * @param args The {@link String} arguments passed in from the command line.
30.      */
31.     public static void main(String[] args) {launch(args);}
32. }
33.
```

DBConnection.java

```
1.  package RMA;
2.
3.  import java.sql.*;
4.
5.  /** DBConnection encapsulates the underlying connectivity and query properties that
6.   *  make up the application's connection to the SQL Server database.
7.   */
8.  class DBConnection {
9.      /** dbURL contains the URL connection {@link String} to connect to the database.
10.      */
11.     private static String dbURL;
12.     /** conn holds a reference to the {@link Connection} object that will be used in
13.      *  database calls.
14.      */
15.     private static Connection conn;
16.     /** stmt maintains a {@link Statement} object reference to use in database queries.
17.      */
18.     private static Statement stmt;
19.     /** cStmt maintains a {@link CallableStatement} object reference to use when executing
20.      *  stored procedures.
21.      */
22.     protected static CallableStatement cStmt;
23.     /** pStmt maintains a {@link PreparedStatement} object reference to use when executing
24.      *  insert, update, or delete statements.
25.      */
26.     protected static PreparedStatement pStmt;
27.     /** rs maintains a {@link ResultSet} object reference to use in database queries.
28.      */
29.     protected static ResultSet rs;
30.
31.
32.     /** Constructor that takes in a username, password, and instance name, and attempts
33.      *  to create a connection to the database.
34.      * @param username The {@link String} username to use to connect to the database.
35.      * @param password The {@link String} password to use to connect to the database.
36.      * @param instance The {@link String} SQL Server instance name to connect.
37.      * @throws SQLException If there is an issue connecting to the database.
38.      */
39.     protected DBConnection(String username, String password, String instance) throws
    SQLException {
40.         dbURL = "jdbc:sqlserver://" + instance + ";" +
41.                 "database=rma;user=" + username + ";password=" + password + ";";
42.         // Create a new JDBC connection.
43.         connect();
44.     }
45.
46.     /** connect creates the database connection.
47.      * @throws SQLException If a database access error occurs.
48.      */
49.     protected void connect() throws SQLException {
50.         // Create a new JDBC connection.
51.         if (conn == null)
52.             conn = DriverManager.getConnection(dbURL);
53.         else if (conn.isClosed())
54.             conn = DriverManager.getConnection(dbURL);
55.     }
56.
57.     /** createCallableStatement prepares the cStmt field with the passed-in executable
    statement.
```

```
58.      * @param sql The query to use to create the {@link CallableStatement}.
59.      * @throws SQLException If a database access error occurs or there is an issue with
60.      * the SQL query.
61.      */
62.     protected void createCallableStatement(String sql) throws SQLException {
63.         if (cStmt != null)
64.             if (!cStmt.isClosed())
65.                 cStmt.close();
66.         if (!isConnected())
67.             connect();
68.         cStmt = conn.prepareCall(sql);
69.     }
70.
71.     /** createPreparedStatement prepares the pStmt field with the passed-in prepared SQL
    statement.
72.      * @param sql The query to use to create the {@link PreparedStatement}.
73.      * @throws SQLException If a database access error occurs or there is an issue with
74.      * the SQL query.
75.      */
76.     protected void createPreparedStatement(String sql) throws SQLException {
77.         if (pStmt != null)
78.             if (!pStmt.isClosed())
79.                 pStmt.close();
80.         if (!isConnected())
81.             connect();
82.         pStmt = conn.prepareStatement(sql);
83.     }
84.
85.     /** isConnected checks that we have a connection to the database.
86.      * @return True if the connection exists, otherwise false.
87.      */
88.     protected boolean isConnected() {
89.         try {
90.             return !conn.isClosed() && (
91.                 (stmt == null ? false : !stmt.isClosed()) ||
92.                 (cStmt == null ? false : !cStmt.isClosed()) ||
93.                 (pStmt == null ? false : !pStmt.isClosed())
94.             );
95.         } catch (SQLException e) {
96.             return false;
97.         }
98.     }
99.
100.      /** executeQuery loads the {@link Statement} instance with the passed-in SQL query
101.       *  and sets the local {@link ResultSet} rs variable with its results.
102.       * @param sql The {@link String} SQL query to parse.
103.       * @throws SQLException If a database access error occurs or there is an issue with
104.       * the SQL query.
105.       */
106.      protected void executeQuery(String sql) throws SQLException {
107.          if (stmt != null)
108.              if (!stmt.isClosed())
109.                  stmt.close();
110.          stmt = conn.createStatement();
111.          rs = stmt.executeQuery(sql);
112.      }
113.
114.      /** executePreparedStatementQuery executes the user-modified pStmt {@link
    PreparedStatement}
115.       *  and sets the local {@link ResultSet} rs variable with its results.
116.       *  @throws SQLException If a database access error occurs or there is an issue with
117.       *  the SQL query.
118.       */
119.      protected void executePreparedStatementQuery() throws SQLException {
120.          rs = pStmt.executeQuery();
```

```
121.      }
122.
123.      /** closeConnection closes the database and statement connections.
124.       * @throws SQLException If a database access error occurs.
125.       */
126.      protected void closeConnection() throws SQLException {
127.          if (stmt != null)
128.              if (!stmt.isClosed())
129.                  stmt.close();
130.          if (cStmt != null)
131.              if (!cStmt.isClosed())
132.                  cStmt.close();
133.          if (pStmt != null)
134.              if (!pStmt.isClosed())
135.                  pStmt.close();
136.          if (conn != null)
137.              if (!conn.isClosed())
138.                  conn.close();
139.      }
140.  }
141.
```

DBService.java

```
1.  package RMA;
2.  import java.sql.SQLException;
3.  import java.sql.Types;
4.  import java.time.LocalDate;
5.  import java.time.LocalDateTime;
6.  import java.util.ArrayList;
7.  import java.util.HashMap;
8.
9.  /** DBService extends the initial core {@link DBConnection} class and
10.  *  implements all logic and functionality for accessing the SQL Server
11.  *  back-end database.
12.  */
13. class DBService extends DBConnection {
14.     /** query contains the most recent {@link String} query used on the database.
15.      */
16.     private static String query;
17.     /** user contains the {@link String} username that was used to log into the database.
18.      */
19.     private static String user;
20.     /** role contains the {@link String} role assigned to the username and password
    credentials used
21.      *  to initially login to the database.
22.      *  (Requirement 1.3)
23.      */
24.     private static String role;
25.     /** instance is a private static instance of this class used to maintain a single point
    of access throughout
26.      *  the application.
27.      */
28.     private static DBService instance;
29.
30.     /** Constructor that takes in a username, password, and instance name, and attempts
31.      *  to create a connection to the database.
32.      * @param username The {@link String} username to use to connect to the database.
33.      * @param password The {@link String} password to use to connect to the database.
34.      * @param instanceName The {@link String} SQL Server instance name to connect.
35.      * @throws SQLException If there is an issue connecting to the database.
36.      */
37.     private DBService(String username, String password, String instanceName) throws
    SQLException {
38.         // First instantiate the superclass, DBConnection.
39.         super(username, password, instanceName);
40.
41.         user = username;
42.         role = getDBRole();
43.     }
44.
45.     /** overloaded getInstance is used to setup the static DBService instance and then
    return it.
46.      * @param username The {@link String} username to use to connect to the database.
47.      * @param password The {@link String} password to use to connect to the database.
48.      * @param instanceName The {@link String} SQL Server instance name to connect.
49.      * @return The static instance of {@link DBService} stored in the {@link DBService}
    class.
50.      * @throws SQLException If there is an issue connecting to the database.
51.      */
52.     public static DBService getInstance(String username, String password, String
    instanceName) throws SQLException {
```

```
53.         if (instance == null)
54.             instance = new DBService(username, password, instanceName);
55.         return instance;
56.     }
57.
58.     /** getInstance returns the static instance stored in this class.
59.      * @return A static instance of DBService.
60.      */
61.     public static DBService getInstance() {return instance;}
62.
63.     /** getUser returns the {@link String} username stored in this DBService.
64.      * @return The {@link String} user field.
65.      */
66.     public String getUser() {return user;}
67.
68.     /** getRole returns the {@link String} role stored in this DBService.
69.      * @return The {@link String} role field.
70.      */
71.     public String getRole() {return role;}
72.
73.     /** getDBRole returns the logged-in user's assigned role in the database as a {@link
    String}.
74.      * @return A {@link String} containing the logged-in user's role.
75.      * @throws SQLException If there is an issue connecting to the database or an issue with
    the SQL query.
76.      */
77.     private String getDBRole() throws SQLException {
78.         // Set up the SQL query.
79.         query = "SELECT DP1.name AS DatabaseRoleName, DP2.name AS DatabaseUserName " +
80.                 "FROM sys.database_role_members AS DRM " +
81.                 "RIGHT OUTER JOIN sys.database_principals AS DP1 " +
82.                 "ON DRM.role_principal_id = DP1.principal_id " +
83.                 "LEFT OUTER JOIN sys.database_principals AS DP2 " +
84.                 "ON DRM.member_principal_id = DP2.principal_id " +
85.                 "WHERE DP2.name = CURRENT_USER " +
86.                 "ORDER BY DatabaseRoleName;";
87.
88.         // Fetch the results.
89.         connect();
90.         executeQuery(query);
91.
92.         String result = "";
93.         if (rs.next())
94.             result = rs.getString("DatabaseRoleName");
95.
96.         // Clean up the connection.
97.         closeConnection();
98.
99.         // Return the result.
100.          return result;
101.     }
102.
103.     /** getCustomerId fetches the customer ID associated with the passed-in customer name.
104.      * @param customerName The {@link String} name of the customer to search for in the
    database.
105.      * @return An int containing the customer's unique id; returns a negative value if the
    customer name was not found.
106.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
107.      */
108.     public int getCustomerId(String customerName) throws SQLException {
109.         // Set up the SQL query.
110.         query = "select customerId from customer where customerName ='" + customerName +
    "';";
111.
```

```
112.          // Fetch the results.
113.          connect();
114.          executeQuery(query);
115.
116.          int result = -1;
117.          if (rs.next())
118.              result = rs.getInt("customerId");
119.
120.          // Clean up the connection.
121.          closeConnection();
122.
123.          // Return the result.
124.          return result;
125.      }
126.
127.      /** getCustomerNames returns an {@link ArrayList} of {@link String} customer names
    stored in the database.
128.       * @return An {@link ArrayList} of {@link String} customer names.
129.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
130.       */
131.      public ArrayList<String> getCustomerNames() throws SQLException {
132.          // Set up the SQL query.
133.          query = "select customerName from customers order by customerName;";
134.
135.          // Fetch the results.
136.          connect();
137.          executeQuery(query);
138.
139.          ArrayList<String> result = new ArrayList<>();
140.          while(rs.next())
141.              result.add(rs.getString("customerName"));
142.
143.          // Clean up the connection.
144.          closeConnection();
145.
146.          // Return the result.
147.          return result;
148.      }
149.
150.      /** getCustomerBusinessNames returns the list of business names associated with a
    customer name through
151.       *  a list of CustomerAddress with showBusinessName explicitly set to true.
152.       * @param customerName The {@link String} name of the customer to search for in the
    database.
153.       * @return An {@link ArrayList} containing {@link CustomerAddress}es, set to only
    output the business name.
154.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
155.       */
156.      public ArrayList<CustomerAddress> getCustomerBusinessNames(String customerName) throws
    SQLException {
157.          // Set up the SQL query.
158.          query = "select ca.addressId, c.customerName, ca.businessName, ca.address1,
    ca.address2," +
159.                  "ca.city, ca.county, ca.stateOrProvince, ca.zip, ca.country, ca.phone,
    ca.fax " +
160.                  "from customerAddresses ca, customers c " +
161.                  "where c.customerId = ca.customerId " +
162.                  "and c.customerName = '" + customerName + "';";
163.
164.          // Fetch the results.
165.          connect();
166.          executeQuery(query);
167.
```

```
168.          ArrayList<CustomerAddress> result = new ArrayList<>();
169.          while (rs.next()) {
170.              CustomerAddress address = new CustomerAddress(
171.                      rs.getInt("addressId"),
172.                      rs.getString("customerName"),
173.                      rs.getString("businessName"),
174.                      rs.getString("address1"),
175.                      rs.getString("address2"),
176.                      rs.getString("city"),
177.                      rs.getString("county"),
178.                      rs.getString("stateOrProvince"),
179.                      rs.getString("zip"),
180.                      rs.getString("country"),
181.                      rs.getString("phone"),
182.                      rs.getString("fax")
183.              );
184.              address.setShowBusinessName(true);
185.              result.add(address);
186.          }
187.
188.          // Clean up the connection.
189.          closeConnection();
190.
191.          // Return the result.
192.          return result;
193.      }
194.
195.      /** getCustomerAddress returns a CustomerAddress containing the address details for
    the given int addressId.
196.       * @param addressId The int ID of the customer address to find.
197.       * @return A CustomerAddress containing the address details.
198.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
199.       */
200.      public CustomerAddress getCustomerAddress(int addressId) throws SQLException {
201.          // Set up the SQL query.
202.          query = "select ca.addressId, c.customerName, ca.businessName, ca.address1,
    ca.address2," +
203.                  "ca.city, ca.county, ca.stateOrProvince, ca.zip, ca.country, ca.phone,
    ca.fax " +
204.                  "from customerAddresses ca, customers c " +
205.                  "where ca.addressId = " + addressId + " " +
206.                  "and c.customerId = ca.customerId;";
207.
208.          // Fetch the results.
209.          connect();
210.          executeQuery(query);
211.
212.          CustomerAddress result = null;
213.          if (rs.next())
214.              result = new CustomerAddress(
215.                  rs.getInt("addressId"),
216.                  rs.getString("customerName"),
217.                  rs.getString("businessName"),
218.                  rs.getString("address1"),
219.                  rs.getString("address2"),
220.                  rs.getString("city"),
221.                  rs.getString("county"),
222.                  rs.getString("stateOrProvince"),
223.                  rs.getString("zip"),
224.                  rs.getString("country"),
225.                  rs.getString("phone"),
226.                  rs.getString("fax")
227.              );
228.
```

```
229.          // Clean up the connection.
230.          closeConnection();
231.
232.          // Return the result.
233.          return result;
234.      }
235.
236.      /** getCustomerAddresses returns an {@link ArrayList} of CustomerAddress instances
    containing the
237.       *  details of each address associated with the specific customer name.
238.       * @param customerName The {@link String} name of the customer to search for in the
    database.
239.       * @return An {@link ArrayList} of type CustomerAddress.
240.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
241.       */
242.      public ArrayList<CustomerAddress> getCustomerAddresses(String customerName) throws
    SQLException {
243.          // Set up the SQL query.
244.          query = "select ca.addressId, c.customerName, ca.businessName, ca.address1,
    ca.address2," +
245.                  "ca.city, ca.county, ca.stateOrProvince, ca.zip, ca.country, ca.phone,
    ca.fax " +
246.                  "from customerAddresses ca, customers c " +
247.                  "where c.customerName = '" + customerName + "' " +
248.                  "and c.customerId = ca.customerId;";
249.
250.          // Fetch the results.
251.          connect();
252.          executeQuery(query);
253.
254.          ArrayList<CustomerAddress> result = new ArrayList<>();
255.          while(rs.next())
256.              result.add(
257.                  new CustomerAddress(
258.                      rs.getInt("addressId"),
259.                      rs.getString("customerName"),
260.                      rs.getString("businessName"),
261.                      rs.getString("address1"),
262.                      rs.getString("address2"),
263.                      rs.getString("city"),
264.                      rs.getString("county"),
265.                      rs.getString("stateOrProvince"),
266.                      rs.getString("zip"),
267.                      rs.getString("country"),
268.                      rs.getString("phone"),
269.                      rs.getString("fax")
270.                  )
271.              );
272.
273.          // Clean up the connection.
274.          closeConnection();
275.
276.          // Return the result.
277.          return result;
278.      }
279.
280.      /** getCustomerAddressPONumbers returns the list of PO numbers associated with the
    given CustomerAddress.
281.       * @param address The CustomerAddress to search for in the database.
282.       * @return An {@link ArrayList} of type {@link String} PO numbers.
283.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
284.       */
```

```
285.      public ArrayList<String> getCustomerAddressPONumbers(CustomerAddress address) throws
    SQLException {
286.          // Set up the SQL query.
287.          query = "select poNumber from purchaseOrders where addressId = '" +
288.                  address.getAddressId()  + "' order by poNumber;";
289.
290.          // Fetch the results.
291.          connect();
292.          executeQuery(query);
293.
294.          ArrayList<String> result = new ArrayList<>();
295.          while(rs.next())
296.              result.add(rs.getString("poNumber"));
297.
298.          // Clean up the connection.
299.          closeConnection();
300.
301.          // Return the result.
302.          return result;
303.      }
304.
305.      /** getPurchaseOrderProduct returns the requested PurchaseOrderProduct whose id is the
    passed-in purchaseOrderProductId.
306.       * @param purchaseOrderProductId The purchaseOrderProductId of the
    PurchaseOrderProduct we wish to fetch.
307.       * @return A PurchaseOrderProduct containing the information for the specified
    purchaseOrderProductId.
308.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
309.       */
310.      public PurchaseOrderProduct getPurchaseOrderProduct(int purchaseOrderProductId) throws
    SQLException {
311.          // Set up the SQL query.
312.          query = "select pop.purchaseOrderProductId, pop.poNumber, p.productName,
    pc.categoryName, " +
313.                  "pop.quantity, pop.orderDate, pop.deliverDate " +
314.                  "from purchaseOrderProducts pop, products p, productCategories pc " +
315.                  "where pop.purchaseOrderProductId = " + purchaseOrderProductId + " " +
316.                  "and pc.categoryId = p.categoryId " +
317.                  "and p.productId = pop.productId " +
318.                  "and p.categoryId = pop.categoryId;";
319.
320.          // Fetch the results.
321.          connect();
322.          executeQuery(query);
323.
324.          PurchaseOrderProduct result = null;
325.          if (rs.next())
326.              result = new PurchaseOrderProduct(
327.                  rs.getInt("purchaseOrderProductId"),
328.                  rs.getString("poNumber"),
329.                  rs.getString("productName"),
330.                  rs.getString("categoryName"),
331.                  rs.getInt("quantity"),
332.                  rs.getObject("orderDate", LocalDate.class),
333.                  rs.getObject("deliverDate", LocalDate.class)
334.              );
335.
336.          // Clean up the connection.
337.          closeConnection();
338.
339.          // Return the result.
340.          return result;
341.      }
342.
```

```
343.      /** getPurchaseOrderProducts returns the {@link ArrayList} of PurchaseOrderProducts
    associated with a
344.       *  given PO number.
345.       * @param poNumber The {@link String} PO number to search for in the database.
346.       * @return An {@link ArrayList} of type PurchaseOrderProduct containing the list of
    products associated
347.       * with the purchase order.
348.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
349.       */
350.      public ArrayList<PurchaseOrderProduct> getPurchaseOrderProducts(String poNumber)
    throws SQLException {
351.          // Set up the SQL query.
352.          query = "select pop.purchaseOrderProductId, pop.poNumber, p.productName,
    pc.categoryName, " +
353.                  "pop.quantity, pop.orderDate, pop.deliverDate " +
354.                  "from purchaseOrderProducts pop, products p, productCategories pc " +
355.                  "where pc.categoryId = p.categoryId " +
356.                  "and p.productId = pop.productId " +
357.                  "and p.categoryId = pop.categoryId " +
358.                  "and pop.poNumber = '" + poNumber + "';";
359.
360.          // Fetch the results.
361.          connect();
362.          executeQuery(query);
363.
364.          ArrayList<PurchaseOrderProduct> result = new ArrayList<>();
365.          while (rs.next())
366.              result.add(
367.                  new PurchaseOrderProduct(
368.                      rs.getInt("purchaseOrderProductId"),
369.                      rs.getString("poNumber"),
370.                      rs.getString("productName"),
371.                      rs.getString("categoryName"),
372.                      rs.getInt("quantity"),
373.                      rs.getObject("orderDate", LocalDate.class),
374.                      rs.getObject("deliverDate", LocalDate.class)
375.                  )
376.              );
377.
378.          // Clean up the connection.
379.          closeConnection();
380.
381.          // Return the result.
382.          return result;
383.      }
384.
385.      /** getRMAStatusId returns the statusId associated with the passed-in status
    description.
386.       * @param description The description of the status for which we want the identifier.
387.       * @return A int containing the value of the statusId; returns a negative value if the
    status was not found.
388.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
389.       */
390.      public int getRMAStatusId(String description) throws SQLException {
391.          // Set up the SQL query.
392.          query = "select statusId from rmaStatuses where description ='" + description +
    "';";
393.
394.          // Fetch the results.
395.          connect();
396.          executeQuery(query);
397.
398.          int result = -1;
```

```
399.          if (rs.next())
400.              result = rs.getInt("statusId");
401.
402.          // Clean up the connection.
403.          closeConnection();
404.
405.          // Return the result.
406.          return result;
407.      }
408.
409.      /** getRMAStatusDescription returns the description associated with an RMA statusId.
410.       * @param statusId The RMA status id int to look up in the database.
411.       * @return The {@link String} description.
412.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
413.       */
414.      public String getRMAStatusDescription(int statusId) throws SQLException {
415.          // Set up the SQL query.
416.          query = "select description from rmaStatuses where statusId =" + statusId + ";";
417.
418.          // Fetch the results.
419.          connect();
420.          executeQuery(query);
421.
422.          String result = "";
423.          if (rs.next())
424.              result = rs.getString("description");
425.
426.          // Clean up the connection.
427.          closeConnection();
428.
429.          // Return the result.
430.          return result;
431.      }
432.
433.      /** getRMAStatuses returns an {@link ArrayList} of type {@link String} containing the
    RMA status
434.       *  descriptions in the database.
435.       * @return An {@link ArrayList} of type {@link String} containing the RMA status
    descriptions.
436.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
437.       */
438.      public ArrayList<String> getRMAStatuses() throws SQLException {
439.          // Set up the SQL query.
440.          query = "select description from rmaStatuses order by description;";
441.
442.          // Fetch the results.
443.          connect();
444.          executeQuery(query);
445.
446.          ArrayList<String> result = new ArrayList<>();
447.          while (rs.next())
448.              result.add(rs.getString("description"));
449.
450.          // Clean up the connection.
451.          closeConnection();
452.
453.          // Return the result.
454.          return result;
455.      }
456.
457.      /** getReturnReasonCodes returns a {@link HashMap} containing types {@link String},
    whose contents
458.       *  are the return reason code initials and the associated description.
```

```
459.       * @return A {@link HashMap} containing types {@link String}, whose contents are
460.       * the return reason code initials and the associated description.
461.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
462.       */
463.      public HashMap<String, String> getReturnReasonCodes() throws SQLException {
464.          // Set up the SQL query.
465.          query = "select returnReasonCode, description from returnReasonCodes order by
    returnReasonCode;";
466.
467.          // Fetch the results.
468.          connect();
469.          executeQuery(query);
470.
471.          HashMap<String, String> result = new HashMap<>();
472.          while (rs.next())
473.              result.put(
474.                  rs.getString("returnReasonCode"),
475.                  rs.getString("description")
476.              );
477.
478.          // Clean up the connection.
479.          closeConnection();
480.
481.          // Return the result.
482.          return result;
483.      }
484.
485.      /** getDispositionId returns the int identifier for the passed-in disposition {@link
    String} description.
486.       * @param disposition The {@link String} disposition description to search for in the
    database.
487.       * @return A int containing the value of the dispositionId; returns a negative value
    if the disposition was not found.
488.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
489.       */
490.      public int getDispositionId(String disposition) throws SQLException {
491.          // Set up the SQL query.
492.          query = "select dispositionId from dispositions where disposition ='" +
    disposition + "';";
493.
494.          // Fetch the results.
495.          connect();
496.          executeQuery(query);
497.
498.          int result = -1;
499.          if (rs.next())
500.              result = rs.getInt("dispositionId");
501.
502.          // Clean up the connection.
503.          closeConnection();
504.
505.          // Return the result.
506.          return result;
507.      }
508.
509.      /** getDispositions returns an {@link ArrayList} of type {@link String}, containing
    the list of available
510.       *  disposition descriptions in the database.
511.       * @return An {@link ArrayList} of type {@link String}, containing the list of
    available disposition
512.       * descriptions in the database.
513.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
```

```
514.       */
515.      public ArrayList<String> getDispositions() throws SQLException {
516.          // Set up the SQL query.
517.          query = "select disposition from dispositions order by disposition;";
518.
519.          // Fetch the results.
520.          connect();
521.          executeQuery(query);
522.
523.          ArrayList<String> result = new ArrayList<>();
524.          while (rs.next())
525.              result.add(rs.getString("disposition"));
526.
527.          // Clean up the connection.
528.          closeConnection();
529.
530.          // Return the result.
531.          return result;
532.      }
533.
534.      /** createRMA creates the initial RMA record in the database using the passed-in
    information.
535.       * @param statusDescription The {@link String} status description to use for this RMA.
536.       * @return The resulting {@link String} RMA ID for this request.
537.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
538.       */
539.      public String createRMA(String statusDescription) throws SQLException {
540.          // First fetch the next RMA ID to use.
541.          query = "{call GetNextRMAId(?)}";
542.
543.          createCallableStatement(query);
544.          cStmt.registerOutParameter(1, Types.VARCHAR);
545.          cStmt.execute();
546.          String rmaId = cStmt.getString(1);
547.          closeConnection();
548.
549.          // Next fetch the statusId to use.
550.          int statusId = getRMAStatusId(statusDescription);
551.
552.          // Now execute the insert statement.
553.          query = "insert into rma" +
554.                  "(rmaId, owner, lastModified, lastModifiedBy, statusId)" +
555.                  "values" +
556.                  "(?, ?, ?, ?, ?);";
557.          connect();
558.          createPreparedStatement(query);
559.          pStmt.setString(1, rmaId);
560.          pStmt.setString(2, getUser());
561.          pStmt.setObject(3, LocalDateTime.now());
562.          pStmt.setString(4, getUser());
563.          pStmt.setInt(5, statusId);
564.          pStmt.executeUpdate();
565.          closeConnection();
566.
567.          return rmaId;
568.      }
569.
570.      /** getRMA fetches the details for the given {@link String} RMA ID and returns it as
    an RMAListViewModel.
571.       * @param rmaId The {@link String} ID of the RMA to fetch.
572.       * @return An RMAListViewModel containing the details of the RMA.
573.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
574.       */
```

```
575.      public RMAListViewModel getRMA(String rmaId) throws SQLException {
576.          // Set up the SQL query.
577.          query = "select r.rmaId, rs.description, rd.shipReplacementRepair, c.customerName, " +
578.                  "ca.businessName, ca.addressId, ca.address1, ca.address2, ca.city, ca.county, " +
579.                  "ca.stateOrProvince, ca.zip, ca.country, ca.phone, ca.fax, pop.purchaseOrderProductId, " +
580.                  "pop.poNumber, p.productName, pc.categoryName, pop.quantity, pop.orderDate, pop.deliverDate, " +
581.                  "rd.returnQuantity, rd.created " +
582.                  "from rma r, rmaDetails rd, rmaStatuses rs, purchaseOrders po, purchaseOrderProducts pop, " +
583.                  "products p, productCategories pc, customerAddresses ca, customers c " +
584.                  "where r.rmaId = '" + rmaId + "' " +
585.                  "and r.statusId = rs.statusId " +
586.                  "and r.rmaId = rd.rmaId " +
587.                  "and rd.poNumber = po.poNumber " +
588.                  "and po.poNumber = pop.poNumber " +
589.                  "and pop.productId = p.productId " +
590.                  "and pop.categoryId = p.categoryId " +
591.                  "and p.categoryId = pc.categoryId " +
592.                  "and po.addressId = ca.addressId " +
593.                  "and ca.customerId = c.customerId;";
594.
595.          // Fetch the results.
596.          connect();
597.          executeQuery(query);
598.
599.          RMAListViewModel result = null;
600.          if (rs.next())
601.              result = new RMAListViewModel(
602.                  rs.getString("rmaId"),
603.                  rs.getString("description"),
604.                  rs.getBoolean("shipReplacementRepair"),
605.                  rs.getString("customerName"),
606.                  rs.getString("businessName"),
607.                  new CustomerAddress(
608.                      rs.getInt("addressId"),
609.                      rs.getString("customerName"),
610.                      rs.getString("businessName"),
611.                      rs.getString("address1"),
612.                      rs.getString("address2"),
613.                      rs.getString("city"),
614.                      rs.getString("county"),
615.                      rs.getString("stateOrProvince"),
616.                      rs.getString("zip"),
617.                      rs.getString("country"),
618.                      rs.getString("phone"),
619.                      rs.getString("fax")
620.                  ),
621.                  new PurchaseOrderProduct(
622.                      rs.getInt("purchaseOrderProductId"),
623.                      rs.getString("poNumber"),
624.                      rs.getString("productName"),
625.                      rs.getString("categoryName"),
626.                      rs.getInt("quantity"),
627.                      rs.getObject("orderDate", LocalDate.class),
628.                      rs.getObject("deliverDate", LocalDate.class)
629.                  ),
630.                  rs.getInt("returnQuantity"),
631.                  false, // shouldDelete
632.                  rs.getObject("created", LocalDateTime.class)
633.              );
634.
```

```
635.          // Clean up the connection.
636.          closeConnection();
637.
638.          // Return the result.
639.          return result;
640.      }
641.
642.      /** getRMAs fetches the list of open RMAs in the database. If the user is an engineer,
643.       *  then the list of RMAs will be all those with an empty engineeringEvaluation in
    RMADetails.
644.       *  In both cases, if a critical RMA exists (RMA that has not been modified for five
    or more days),
645.       *  then the list will only show critical RMAs.
646.       * @return An {@link ArrayList} of RMAListViewModel containing the RMAs' details.
647.       * @throws SQLException If there is an issue connecting to the database.
648.       */
649.      public ArrayList<RMAListViewModel> getRMAs() throws SQLException {
650.          // Create the result list for later use.
651.          ArrayList<RMAListViewModel> result = new ArrayList<>();
652.
653.          // First check if there are any critical RMAs.
654.          if (!getRole().equals("engineer"))
655.              query = "select r.rmaId, rs.description, rd.shipReplacementRepair,
    c.customerName, " +
656.                      "ca.businessName, ca.addressId, ca.address1, ca.address2, ca.city,
    ca.county, " +
657.                      "ca.stateOrProvince, ca.zip, ca.country, ca.phone, ca.fax,
    pop.purchaseOrderProductId, " +
658.                      "pop.poNumber, p.productName, pc.categoryName, pop.quantity,
    pop.orderDate, pop.deliverDate, " +
659.                      "rd.returnQuantity, rd.created " +
660.                  "from rma r, rmaDetails rd, rmaStatuses rs, purchaseOrders po,
    purchaseOrderProducts pop, " +
661.                       "products p, productCategories pc, customerAddresses ca,
    customers c " +
662.                  "where rs.description != 'Closed' " +
663.                      "and datediff(day, r.lastModified, ?) >= 5 " +
664.                      "and rd.purchaseOrderProductId = pop.purchaseOrderProductId " +
665.                      "and r.statusId = rs.statusId " +
666.                      "and r.rmaId = rd.rmaId " +
667.                      "and rd.poNumber = po.poNumber " +
668.                      "and po.poNumber = pop.poNumber " +
669.                      "and pop.productId = p.productId " +
670.                      "and pop.categoryId = p.categoryId " +
671.                      "and p.categoryId = pc.categoryId " +
672.                      "and po.addressId = ca.addressId " +
673.                      "and ca.customerId = c.customerId " +
674.                  "order by r.rmaId;";
675.          else
676.              query = "select r.rmaId, rs.description, rd.shipReplacementRepair,
    c.customerName, " +
677.                      "ca.businessName, ca.addressId, ca.address1, ca.address2, ca.city,
    ca.county, " +
678.                      "ca.stateOrProvince, ca.zip, ca.country, ca.phone, ca.fax,
    pop.purchaseOrderProductId, " +
679.                      "pop.poNumber, p.productName, pc.categoryName, pop.quantity,
    pop.orderDate, pop.deliverDate, " +
680.                      "rd.returnQuantity, rd.created " +
681.                  "from rma r, rmaDetails rd, rmaStatuses rs, purchaseOrders po,
    purchaseOrderProducts pop, " +
682.                  "products p, productCategories pc, customerAddresses ca, customers c "
    +
683.                  "where rs.description != 'Closed' " +
684.                      "and datediff(day, r.lastModified, ?) >= 5 " +
685.                      "and rd.purchaseOrderProductId = pop.purchaseOrderProductId " +
```

```
                            "and rd.engineeringEvaluation = '' " +
                            "and r.statusId = rs.statusId " +
                            "and r.rmaId = rd.rmaId " +
                            "and rd.poNumber = po.poNumber " +
                            "and po.poNumber = pop.poNumber " +
                            "and pop.productId = p.productId " +
                            "and pop.categoryId = p.categoryId " +
                            "and p.categoryId = pc.categoryId " +
                            "and po.addressId = ca.addressId " +
                            "and ca.customerId = c.customerId " +
                        "order by r.rmaId;";
            connect();
            createPreparedStatement(query);
            pStmt.setObject(1, LocalDateTime.now());
            executePreparedStatementQuery();
            if (rs.next()) { // We have critical RMAs to process.
                do {
                    result.add(
                        new RMAListViewModel(
                            rs.getString("rmaId"),
                            rs.getString("description"),
                            rs.getBoolean("shipReplacementRepair"),
                            rs.getString("customerName"),
                            rs.getString("businessName"),
                            new CustomerAddress(
                                rs.getInt("addressId"),
                                rs.getString("customerName"),
                                rs.getString("businessName"),
                                rs.getString("address1"),
                                rs.getString("address2"),
                                rs.getString("city"),
                                rs.getString("county"),
                                rs.getString("stateOrProvince"),
                                rs.getString("zip"),
                                rs.getString("country"),
                                rs.getString("phone"),
                                rs.getString("fax")
                            ),
                            new PurchaseOrderProduct(
                                rs.getInt("purchaseOrderProductId"),
                                rs.getString("poNumber"),
                                rs.getString("productName"),
                                rs.getString("categoryName"),
                                rs.getInt("quantity"),
                                rs.getObject("orderDate", LocalDate.class),
                                rs.getObject("deliverDate", LocalDate.class)
                            ),
                            rs.getInt("returnQuantity"),
                     false, // shouldDelete
                            rs.getObject("created", LocalDateTime.class)
                        )
                    );
                } while (rs.next());

            } else { // Check if we have non-critical RMAs to process.
                // Reset the connection and statements.
                closeConnection();

                // Queries for non-critical RMAs.
                if (!getRole().equals("engineer"))
                    query = "select r.rmaId, rs.description, rd.shipReplacementRepair,
    c.customerName, " +
                            "ca.businessName, ca.addressId, ca.address1, ca.address2, ca.city,
    ca.county, " +
```

```
748.                         "ca.stateOrProvince, ca.zip, ca.country, ca.phone, ca.fax,
    pop.purchaseOrderProductId, " +
749.                         "pop.poNumber, p.productName, pc.categoryName, pop.quantity,
    pop.orderDate, pop.deliverDate, " +
750.                         "rd.returnQuantity, rd.created " +
751.                         "from rma r, rmaDetails rd, rmaStatuses rs, purchaseOrders po,
    purchaseOrderProducts pop, " +
752.                         "products p, productCategories pc, customerAddresses ca, customers
    c " +
753.                         "where rs.description != 'Closed' " +
754.                         "and datediff(day, r.lastModified, ?) < 5 " +
755.                         "and rd.purchaseOrderProductId = pop.purchaseOrderProductId " +
756.                         "and r.statusId = rs.statusId " +
757.                         "and r.rmaId = rd.rmaId " +
758.                         "and rd.poNumber = po.poNumber " +
759.                         "and po.poNumber = pop.poNumber " +
760.                         "and pop.productId = p.productId " +
761.                         "and pop.categoryId = p.categoryId " +
762.                         "and p.categoryId = pc.categoryId " +
763.                         "and po.addressId = ca.addressId " +
764.                         "and ca.customerId = c.customerId " +
765.                         "order by r.rmaId;";
766.                 else
767.                     query = "select r.rmaId, rs.description, rd.shipReplacementRepair,
    c.customerName, " +
768.                         "ca.businessName, ca.addressId, ca.address1, ca.address2, ca.city,
    ca.county, " +
769.                         "ca.stateOrProvince, ca.zip, ca.country, ca.phone, ca.fax,
    pop.purchaseOrderProductId, " +
770.                         "pop.poNumber, p.productName, pc.categoryName, pop.quantity,
    pop.orderDate, pop.deliverDate, " +
771.                         "rd.returnQuantity, rd.created " +
772.                         "from rma r, rmaDetails rd, rmaStatuses rs, purchaseOrders po,
    purchaseOrderProducts pop, " +
773.                         "products p, productCategories pc, customerAddresses ca, customers
    c " +
774.                         "where rs.description != 'Closed' " +
775.                         "and datediff(day, r.lastModified, ?) < 5 " +
776.                         "and rd.purchaseOrderProductId = pop.purchaseOrderProductId " +
777.                         "and rd.engineeringEvaluation = '' " +
778.                         "and r.statusId = rs.statusId " +
779.                         "and r.rmaId = rd.rmaId " +
780.                         "and rd.poNumber = po.poNumber " +
781.                         "and po.poNumber = pop.poNumber " +
782.                         "and pop.productId = p.productId " +
783.                         "and pop.categoryId = p.categoryId " +
784.                         "and p.categoryId = pc.categoryId " +
785.                         "and po.addressId = ca.addressId " +
786.                         "and ca.customerId = c.customerId " +
787.                         "order by r.rmaId;";
788.
789.                 connect();
790.                 createPreparedStatement(query);
791.                 pStmt.setObject(1, LocalDateTime.now());
792.                 executePreparedStatementQuery();
793.                 if (rs.next()) // We have non-critical RMAs to process.
794.                     do {
795.                         result.add(
796.                             new RMAListViewModel(
797.                                 rs.getString("rmaId"),
798.                                 rs.getString("description"),
799.                                 rs.getBoolean("shipReplacementRepair"),
800.                                 rs.getString("customerName"),
801.                                 rs.getString("businessName"),
802.                                 new CustomerAddress(
```

```
803.                                 rs.getInt("addressId"),
804.                                 rs.getString("customerName"),
805.                                 rs.getString("businessName"),
806.                                 rs.getString("address1"),
807.                                 rs.getString("address2"),
808.                                 rs.getString("city"),
809.                                 rs.getString("county"),
810.                                 rs.getString("stateOrProvince"),
811.                                 rs.getString("zip"),
812.                                 rs.getString("country"),
813.                                 rs.getString("phone"),
814.                                 rs.getString("fax")
815.                             ),
816.                             new PurchaseOrderProduct(
817.                                 rs.getInt("purchaseOrderProductId"),
818.                                 rs.getString("poNumber"),
819.                                 rs.getString("productName"),
820.                                 rs.getString("categoryName"),
821.                                 rs.getInt("quantity"),
822.                                 rs.getObject("orderDate", LocalDate.class),
823.                                 rs.getObject("deliverDate", LocalDate.class)
824.                             ),
825.                             rs.getInt("returnQuantity"),
826.                             false, // shouldDelete
827.                             rs.getObject("created", LocalDateTime.class)
828.                         )
829.                     );
830.                 } while (rs.next());
831.         }
832.
833.         // Clean up connection.
834.         closeConnection();
835.
836.         // Return the RMAs.
837.         return result;
838.     }
839.
840.     /** createRMADetails creates the rmaDetails record associated with the initial RMA
    record using the passed-in information.
841.      * @param rmaId The {@link String} ID of the RMA whose details we are creating.
842.      * @param returnReasonCode The {@link String} Return Reason Code we are assigning to
    this RMA.
843.      * @param creditReplaceRepair Whether we are deciding to credit, replace, or repair
    the returned items, saved as a {@link String}.
844.      * @param purchaseOrderProductId The unique int identifier of the product being
    returned by the customer.
845.      * @param returnQuantity The int amount of product being returned by the customer.
846.      * @param returnLabelTracker The {@link String} return label tracking number used to
    ship the product to us.
847.      * @param additionalInfo Any additional info needed to process the RMA, entered in by
    an Analyst. Stored as a {@link String}.
848.      * @param poNumber The {@link String} purchase order number being referenced in this
    RMA.
849.      * @param initialEvaluation The {@link String} initial evaluation by an Analyst of the
    product's condition and the RMA request.
850.      * @param engineeringEvaluation The {@link String} evaluation of the returned product
    by an Engineer.
851.      * @param disposition The {@link String} disposition to assign to the returned product
    by an Analyst.
852.      * @param dispositionNotes Any additional notes on the disposition written by an
    Analyst. Stored as a {@link String}.
853.      * @param replacementTrackingNumber The {@link String} tracking number for the
    replacement being sent back to the customer.
854.      * @param replacementShipDate The {@link LocalDate} date the replacement was or will
    be shipped back to the customer.
```

```
855.       * @param shipReplacementRepair A boolean indicating whether or not we will be
    shipping back a replacement or repair to the customer.
856.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
857.       */
858.      public void createRMADetails (
859.              String rmaId, String returnReasonCode, String creditReplaceRepair, int
    purchaseOrderProductId, int returnQuantity,
860.              String returnLabelTracker, String additionalInfo, String poNumber, String
    initialEvaluation, String engineeringEvaluation,
861.              String disposition, String dispositionNotes, String replacementTrackingNumber,
    LocalDate replacementShipDate,
862.              boolean shipReplacementRepair
863.      ) throws SQLException {
864.          // First setup the SQL query to fetch the dispositionId from the database.
865.          query = "select dispositionId from dispositions where disposition = '" +
    disposition + "';";
866.          // Connect to the database and execute the query.
867.          connect();
868.          executeQuery(query);
869.
870.          Integer dispositionId = null;
871.          if (rs.next())
872.              dispositionId = rs.getInt("dispositionId");
873.
874.          // Setup the SQL query to insert a new RMADetails record into the database.
875.          query = "insert into rmaDetails(" +
876.                  "rmaId, created, createdBy, returnReasonCode, creditReplaceRepair,
    purchaseOrderProductId, returnQuantity," +
877.                  "returnLabelTracker, additionalInfo, poNumber, initialEvaluation,
    engineeringEvaluation, dispositionId, " +
878.                  "dispositionNotes, replacementTrackingNumber, replacementShipDate,
    shipReplacementRepair" +
879.                  ") values (" +
880.                  "?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?" +
881.                  ");";
882.
883.          // Create a new PreparedStatement and execute the query.
884.          createPreparedStatement(query);
885.          pStmt.setString(1, rmaId);
886.          pStmt.setObject(2, LocalDateTime.now());
887.          pStmt.setString(3, getUser());
888.          pStmt.setString(4, returnReasonCode);
889.          pStmt.setString(5, creditReplaceRepair);
890.          pStmt.setInt(6, purchaseOrderProductId);
891.          pStmt.setInt(7, returnQuantity);
892.          pStmt.setString(8, returnLabelTracker);
893.          pStmt.setString(9, additionalInfo);
894.          pStmt.setString(10, poNumber);
895.          pStmt.setString(11, initialEvaluation);
896.          pStmt.setString(12, engineeringEvaluation);
897.          if (dispositionId != null)
898.              pStmt.setInt(13, dispositionId);
899.          else
900.              pStmt.setNull(13, Types.INTEGER);
901.          pStmt.setString(14, dispositionNotes);
902.          pStmt.setString(15, replacementTrackingNumber);
903.          pStmt.setObject(16, replacementShipDate);
904.          pStmt.setBoolean(17, shipReplacementRepair);
905.          pStmt.executeUpdate();
906.          closeConnection();
907.      }
908.
909.      /** getRMACustomerName returns the customer name associated with an RMA ID, through
    its PO number and the PO number's
```

```
910.        *  associated address ID.
911.        * @param rmaId The RMA whose customer name we wish to look up.
912.        * @return A {@link String} containing the name of the customer associated with the PO
    and address in the RMA.
913.        * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
914.        */
915.       public String getRMACustomerName(String rmaId) throws SQLException {
916.           // Set up the SQL query.
917.           query = "select c.customerName " +
918.                   "from customers c, customerAddresses ca, purchaseOrders po, rmaDetails rd
    " +
919.                   "where rd.rmaId ='" + rmaId + "' " +
920.                   "and rd.poNumber = po.poNumber " +
921.                   "and po.addressId = ca.addressId " +
922.                   "and ca.customerId = c.customerId;";
923.
924.           // Fetch the results.
925.           connect();
926.           executeQuery(query);
927.
928.           String result = "";
929.           if (rs.next())
930.               result = rs.getString("customerName");
931.
932.           // Clean up the connection.
933.           closeConnection();
934.
935.           // Return the result.
936.           return result;
937.       }
938.
939.       /** getRMABusinessName returns a {@link CustomerAddress} set to show the business name
    associated with the RMA whose
940.        *  ID we are searching, through its PO number and the PO number's associated address
    ID.
941.        *  @param rmaId The RMA whose business name we wish to look up.
942.        *  @return A {@link CustomerAddress} set to show the name of the business associated
    with the PO and address in the RMA.
943.        *  @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
944.        */
945.       public CustomerAddress getRMABusinessName(String rmaId) throws SQLException {
946.           // Set up the SQL query.
947.           query = "select ca.addressId, c.customerName, ca.businessName, ca.address1,
    ca.address2, " +
948.                   "ca.city, ca.county, ca.stateOrProvince, ca.zip, ca.country, ca.phone,
    ca.fax " +
949.                   "from rmaDetails rd, purchaseOrders PO, customerAddresses ca, customers c
    " +
950.                   "where rd.rmaId = '" + rmaId + "' " +
951.                   "and rd.poNumber = po.poNumber " +
952.                   "and po.addressId = ca.addressId " +
953.                   "and ca.customerId = c.customerId;";
954.
955.           // Fetch the results.
956.           connect();
957.           executeQuery(query);
958.
959.           CustomerAddress result = null;
960.           if (rs.next()) {
961.               result = new CustomerAddress(
962.                   rs.getInt("addressId"),
963.                   rs.getString("customerName"),
964.                   rs.getString("businessName"),
```

```
965.                    rs.getString("address1"),
966.                    rs.getString("address2"),
967.                    rs.getString("city"),
968.                    rs.getString("county"),
969.                    rs.getString("stateOrProvince"),
970.                    rs.getString("zip"),
971.                    rs.getString("country"),
972.                    rs.getString("phone"),
973.                    rs.getString("fax")
974.                );
975.                result.setShowBusinessName(true);
976.            }
977.
978.            // Clean up the connection.
979.            closeConnection();
980.
981.            // Return the result.
982.            return result;
983.        }
984.
985.        /** getRMAPONumber returns the PO number associated with the given {@link String} RMA
    ID.
986.         *  @param rmaId The RMA whose PO number we wish to look up.
987.         *  @return A {@link String} containing the PO number associated with the RMA.
988.         *  @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
989.         */
990.        public String getRMAPONumber(String rmaId) throws SQLException {
991.            // Set up the SQL query.
992.            query = "select poNumber from rmaDetails where rmaId ='" + rmaId + "';";
993.
994.            // Fetch the results.
995.            connect();
996.            executeQuery(query);
997.
998.            String result = "";
999.            if (rs.next())
1000.                result = rs.getString("poNumber");
1001.
1002.            // Clean up the connection.
1003.            closeConnection();
1004.
1005.            // Return the result.
1006.            return result;
1007.        }
1008.
1009.        /** getRMAAddress returns the address information associated with the {@link String}
    RMA ID's PO number in a CustomerAddress.
1010.         * @param rmaId The RMA whose address information we wish to look up.
1011.         * @return A CustomerAddress containing all the address information associated with
    the RMA's PO number.
1012.         * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1013.         */
1014.        public CustomerAddress getRMAAddress(String rmaId) throws SQLException {
1015.            // Set up the SQL query.
1016.            query = "select ca.addressId, c.customerName, ca.businessName, ca.address1,
    ca.address2, " +
1017.                    "ca.city, ca.county, ca.stateOrProvince, ca.zip, ca.country, ca.phone,
    ca.fax " +
1018.                    "from rmaDetails rd, purchaseOrders PO, customerAddresses ca, customers c
    " +
1019.                    "where rd.rmaId = '" + rmaId + "' " +
1020.                    "and rd.poNumber = po.poNumber " +
1021.                    "and po.addressId = ca.addressId " +
```

```
1022.                 "and ca.customerId = c.customerId;";
1023.
1024.         // Fetch the results.
1025.         connect();
1026.         executeQuery(query);
1027.
1028.         CustomerAddress result = null;
1029.         if (rs.next())
1030.             result = new CustomerAddress(
1031.                 rs.getInt("addressId"),
1032.                 rs.getString("customerName"),
1033.                 rs.getString("businessName"),
1034.                 rs.getString("address1"),
1035.                 rs.getString("address2"),
1036.                 rs.getString("city"),
1037.                 rs.getString("county"),
1038.                 rs.getString("stateOrProvince"),
1039.                 rs.getString("zip"),
1040.                 rs.getString("country"),
1041.                 rs.getString("phone"),
1042.                 rs.getString("fax")
1043.             );
1044.
1045.         // Clean up the connection.
1046.         closeConnection();
1047.
1048.         // Return the result.
1049.         return result;
1050.     }
1051.
1052.     /** getOwners returns an {@link ArrayList} of usernames stored in the database.
1053.      * @return An {@link ArrayList} of type {@link String} containing the list of
    usernames in the RMA database.
1054.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1055.      */
1056.     public ArrayList<String> getOwners() throws SQLException {
1057.         // Set up the SQL query.
1058.         query = "select name " +
1059.                 "from sys.database_principals " +
1060.                 "where type not in ('A', 'G', 'R', 'X') " +
1061.                 "and sid is not null " +
1062.                 "and name not in ('guest', 'dbo') " +
1063.                 "order by name;";
1064.
1065.         // Fetch the results.
1066.         connect();
1067.         executeQuery(query);
1068.
1069.         ArrayList<String> result = new ArrayList<>();
1070.         while (rs.next())
1071.             result.add(rs.getString("name"));
1072.
1073.         // Clean up the connection.
1074.         closeConnection();
1075.
1076.         // Return the result.
1077.         return result;
1078.     }
1079.
1080.     /** getRMAOwner returns the {@link String} username of the owner stored in the
    database for the specified {@link String} RMA ID.
1081.      * @param rmaId The {@link String} whose owner we wish to fetch.
1082.      * @return A {@link String} containing the name of the current owner.
```

```
1083.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1084.      */
1085.     public String getRMAOwner(String rmaId) throws SQLException {
1086.         // Set up the SQL query.
1087.         query = "select owner from [rma] where rmaId ='" + rmaId + "';";
1088.
1089.         // Fetch the results.
1090.         connect();
1091.         executeQuery(query);
1092.
1093.         String result = "";
1094.         if (rs.next())
1095.             result = rs.getString("owner");
1096.
1097.         // Clean up the connection.
1098.         closeConnection();
1099.
1100.         // Return the result.
1101.         return result;
1102.     }
1103.
1104.     /** updateRMAOwner updates the RMA owner in the database for the given RMA ID with the
    passed-in username.
1105.      * @param rmaId The {@link String} ID of the RMA to update.
1106.      * @param newOwner The {@link String} new owner name to associate with the RMA.
1107.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1108.      */
1109.     public void updateRMAOwner(String rmaId, String newOwner) throws SQLException {
1110.         // Setup the SQL query to update the database.
1111.         query = "update rma " +
1112.                 "set owner = ? " +
1113.                 "where rmaId = ?;";
1114.
1115.         // Connect to the database and execute the query.
1116.         connect();
1117.         createPreparedStatement(query);
1118.         pStmt.setString(2, rmaId);
1119.         pStmt.setString(1, newOwner);
1120.         pStmt.executeUpdate();
1121.         closeConnection();
1122.     }
1123.
1124.     /** getRMALastModified fetches the {@link LocalDateTime} datetime stored in the
    database for the given {@link String} RMA ID.
1125.      * @param rmaId The {@link String} ID of the RMA whose last modified datetime we wish
    to fetch.
1126.      * @return A {@link LocalDateTime} containing the date and time of the last time the
    RMA was modified.
1127.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1128.      */
1129.     public LocalDateTime getRMALastModified(String rmaId) throws SQLException {
1130.         // Set up the SQL query.
1131.         query = "select lastModified from rma where rmaId ='" + rmaId + "';";
1132.
1133.         // Fetch the results.
1134.         connect();
1135.         executeQuery(query);
1136.
1137.         LocalDateTime result = null;
1138.         if (rs.next())
1139.             result = rs.getObject("lastModified", LocalDateTime.class);
1140.
```

```
1141.         // Clean up the connection.
1142.         closeConnection();
1143.
1144.         // Return the result.
1145.         return result;
1146.     }
1147.
1148.     /** updateRMALastModified updates the {@link LocalDateTime} date and time of the RMA's
    lastModified column to now.
1149.      * @param rmaId The {@link String} ID of the RMA to update.
1150.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1151.      */
1152.     public void updateRMALastModified(String rmaId) throws SQLException {
1153.         // Setup the SQL query to update the database.
1154.         query = "update rma " +
1155.                 "set lastModified = ? " +
1156.                 "where rmaId = ?;";
1157.
1158.         // Connect to the database and execute the query.
1159.         connect();
1160.         createPreparedStatement(query);
1161.         pStmt.setString(2, rmaId);
1162.         pStmt.setObject(1, LocalDateTime.now());
1163.         pStmt.executeUpdate();
1164.         closeConnection();
1165.     }
1166.
1167.     /** getRMALastModifiedBy returns the {@link String} username of the user that last
    modified the RMA with the given {@link String} ID.
1168.      * @param rmaId The {@link String} ID of the RMA whose last modified username we wish
    to fetch.
1169.      * @return A {@link String} containing the username that last modified the RMA.
1170.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1171.      */
1172.     public String getRMALastModifiedBy(String rmaId) throws SQLException {
1173.         // Set up the SQL query.
1174.         query = "select lastModifiedBy from rma where rmaId ='" + rmaId + "';";
1175.
1176.         // Fetch the results.
1177.         connect();
1178.         executeQuery(query);
1179.
1180.         String result = "";
1181.         if (rs.next())
1182.             result = rs.getString("lastModifiedBy");
1183.
1184.         // Clean up the connection.
1185.         closeConnection();
1186.
1187.         // Return the result.
1188.         return result;
1189.     }
1190.
1191.     /** updateRMALastModifiedBy updates the {@link String} username in the lastModifiedBy
    column of the RMA with the given ID to
1192.      *  the current logged-in user.
1193.      * @param rmaId The {@link String} ID of the RMA to update.
1194.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1195.      */
1196.     public void updateRMALastModifiedBy(String rmaId) throws SQLException {
1197.         // Setup the SQL query to update the database.
1198.         query = "update rma " +
```

```
1199.                 "set lastModifiedBy = ? " +
1200.                 "where rmaId = ?;";
1201.
1202.         // Connect to the database and execute the query.
1203.         connect();
1204.         createPreparedStatement(query);
1205.         pStmt.setString(2, rmaId);
1206.         pStmt.setString(1, getUser());
1207.         pStmt.executeUpdate();
1208.         closeConnection();
1209.     }
1210.
1211.     /** getRMAStatus returns the {@link String} status description stored in the database
    for the given {@link String} RMA ID.
1212.      * @param rmaId The {@link String} ID of the RMA whose status description we wish to
    fetch.
1213.      * @return A {@link String} containing the RMA's status description.
1214.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1215.      */
1216.     public String getRMAStatus(String rmaId) throws SQLException {
1217.         // Set up the SQL query.
1218.         query = "select rs.description " +
1219.                 "from rma r, rmaStatuses rs " +
1220.                 "where r.rmaId = '" + rmaId + "' " +
1221.                 "and r.statusId = rs.statusId;";
1222.
1223.         // Fetch the results.
1224.         connect();
1225.         executeQuery(query);
1226.
1227.         String result = "";
1228.         if (rs.next())
1229.             result = rs.getString("description");
1230.
1231.         // Clean up the connection.
1232.         closeConnection();
1233.
1234.         // Return the result.
1235.         return result;
1236.     }
1237.
1238.     /** updateRMAStatus updates the status assigned to the RMA with the given {@link
    String} ID to the passed-in
1239.      *  {@link String} description.
1240.      * @param rmaId The {@link String} ID of the RMA to update.
1241.      * @param description The {@link String} status description whose identifier we want
    to use to update the RMA.
1242.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1243.      */
1244.     public void updateRMAStatus(String rmaId, String description) throws SQLException {
1245.         // Setup the SQL query to update the database.
1246.         query = "update rma " +
1247.                 "set statusId = (select statusId from rmaStatuses where description = ?) "
    +
1248.                 "where rmaId = ?;";
1249.
1250.         // Connect to the database and execute the query.
1251.         connect();
1252.         createPreparedStatement(query);
1253.         pStmt.setString(2, rmaId);
1254.         pStmt.setString(1, description);
1255.         pStmt.executeUpdate();
1256.         closeConnection();
```

```
1257.     }
1258.
1259.     /** getRMACreditReplaceRepair returns the current value (credit, replace, or repair)
    stored in the database for the
1260.      *  given {@link String} RMA ID.
1261.      *  @param rmaId The {@link String} ID of the RMA whose creditReplaceRepair value we
    wish to fetch.
1262.      *  @return A {@link String} containing the RMA's creditReplaceRepair value.
1263.      *  @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1264.      */
1265.     public String getRMACreditReplaceRepair(String rmaId) throws SQLException {
1266.         // Set up the SQL query.
1267.         query = "select creditReplaceRepair from rmaDetails where rmaId =  '" + rmaId +
    "';";
1268.
1269.         // Fetch the results.
1270.         connect();
1271.         executeQuery(query);
1272.
1273.         String result = "";
1274.         if (rs.next())
1275.             result = rs.getString("creditReplaceRepair");
1276.
1277.         // Clean up the connection.
1278.         closeConnection();
1279.
1280.         // Return the result.
1281.         return result;
1282.     }
1283.
1284.     /** updateRMACreditReplaceRepair updates the specified RMA with the passed-in credit,
    replace, or repair value.
1285.      * @param rmaId The {@link String} ID of the RMA whose details we want to update.
1286.      * @param creditReplaceRepair The {@link String} new value of credit, replace, or
    repair that we want to save in the database.
1287.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1288.      */
1289.     public void updateRMACreditReplaceRepair(String rmaId, String creditReplaceRepair)
    throws SQLException {
1290.         // Setup the SQL query to update the database.
1291.         query = "update rmaDetails " +
1292.                 "set creditReplaceRepair = ? " +
1293.                 "where rmaId = ?;";
1294.
1295.         // Connect to the database and execute the query.
1296.         connect();
1297.         createPreparedStatement(query);
1298.         pStmt.setString(2, rmaId);
1299.         pStmt.setString(1, creditReplaceRepair);
1300.         pStmt.executeUpdate();
1301.         closeConnection();
1302.     }
1303.
1304.     /** getRMAReturnReasonCode returns the current return reason code and its description
    associated with the given
1305.      *  {@link String} RMA ID.
1306.      * @param rmaId The {@link String} ID of the RMA whose return reason code description
    we wish to fetch.
1307.      * @return A {@link String} containing the return reason code and its description,
    separated by a space, and
1308.      *         associated with the given RMA ID.
1309.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
```

```
1310.      */
1311.     public String getRMAReturnReasonCode(String rmaId) throws SQLException {
1312.         // Set up the SQL query.
1313.         query = "select rrc.returnReasonCode, rrc.description " +
1314.                 "from rmaDetails rd, returnReasonCodes rrc " +
1315.                 "where rd.rmaId = '" + rmaId + "' " +
1316.                 "and rd.returnReasonCode = rrc.returnReasonCode;";
1317.
1318.         // Fetch the results.
1319.         connect();
1320.         executeQuery(query);
1321.
1322.         String result = "";
1323.         if (rs.next())
1324.             result = rs.getString("returnReasonCode") + " " + rs.getString("description");
1325.
1326.         // Clean up the connection.
1327.         closeConnection();
1328.
1329.         // Return the result.
1330.         return result;
1331.     }
1332.
1333.     /** updateRMAReturnReasonCode updates the specified RMA with the passed-in
    returnReasonCodeDescription ("Code Description").
1334.      * @param rmaId The {@link String} ID of the RMA whose details we want to update.
1335.      * @param returnReasonCodeDescription The {@link String} ("Code Description") for the
    return reason code we want to use in RMA.
1336.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1337.      */
1338.     public void updateRMAReturnReasonCode(String rmaId, String
    returnReasonCodeDescription) throws SQLException {
1339.         // Setup the SQL query to update the database.
1340.         query = "update rmaDetails " +
1341.                 "set returnReasonCode = ? " +
1342.                 "where rmaId = ?;";
1343.
1344.         // Connect to the database and execute the query.
1345.         connect();
1346.         createPreparedStatement(query);
1347.         pStmt.setString(2, rmaId);
1348.         pStmt.setString(1, returnReasonCodeDescription);
1349.         pStmt.executeUpdate();
1350.         closeConnection();
1351.     }
1352.
1353.     /** getRMAAdditionalInfo returns a {@link String} containing the additional info notes
    stored in the RMA Details for
1354.      *  the given {@link String} RMA ID.
1355.      *  @param rmaId The {@link String} ID of the RMA whose return additional info notes
    we wish to fetch.
1356.      *  @return A {@link String} containing the additional info notes associated with the
    given RMA ID.
1357.      *  @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1358.      */
1359.     public String getRMAAdditionalInfo(String rmaId) throws SQLException {
1360.         // Set up the SQL query.
1361.         query = "select additionalInfo from rmaDetails where rmaId =  '" + rmaId + "';";
1362.
1363.         // Fetch the results.
1364.         connect();
1365.         executeQuery(query);
1366.
```

```
1367.         String result = "";
1368.         if (rs.next())
1369.             result = rs.getString("additionalInfo");
1370.
1371.         // Clean up the connection.
1372.         closeConnection();
1373.
1374.         // Return the result.
1375.         return result;
1376.     }
1377.
1378.     /** updateRMAAdditionalInfo updates the requested RMA's details with the given {@link
    String} ID with the updated
1379.      *  information passed-in through the {@link String }additionalInfo parameter.
1380.      * @param rmaId The {@link String} ID of the RMA to update.
1381.      * @param additionalInfo The updated {@link String} Additional Info value to store in
    the database.
1382.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1383.      */
1384.     public void updateRMAAdditionalInfo(String rmaId, String additionalInfo) throws
    SQLException {
1385.         // Setup the SQL query to update the database.
1386.         query = "update rmaDetails " +
1387.                 "set additionalInfo = ? " +
1388.                 "where rmaId = ?;";
1389.
1390.         // Connect to the database and execute the query.
1391.         connect();
1392.         createPreparedStatement(query);
1393.         pStmt.setString(2, rmaId);
1394.         pStmt.setString(1, additionalInfo);
1395.         pStmt.executeUpdate();
1396.         closeConnection();
1397.     }
1398.
1399.     /** getRMACreated returns a {@link LocalDateTime} object containing the creation date
    and time of the given RMA.
1400.      * @param rmaId The {@link String} ID of the RMA whose created datetime we wish to
    fetch.
1401.      * @return A {@link LocalDateTime} containing the creation date and time of the given
    RMA ID.
1402.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1403.      */
1404.     public LocalDateTime getRMACreated(String rmaId) throws SQLException {
1405.         // Set up the SQL query.
1406.         query = "select created from rmaDetails where rmaId =  '" + rmaId + "';";
1407.
1408.         // Fetch the results.
1409.         connect();
1410.         executeQuery(query);
1411.
1412.         LocalDateTime result = null;
1413.         if (rs.next())
1414.             result = rs.getObject("created", LocalDateTime.class);
1415.
1416.         // Clean up the connection.
1417.         closeConnection();
1418.
1419.         // Return the result.
1420.         return result;
1421.     }
1422.
```

```
1423.     /** getRMACreatedBy returns a {@link String} containing the username that created the
    given RMA request.
1424.      * @param rmaId The {@link String} ID of the RMA whose created username we wish to
    fetch.
1425.      * @return A {@link String} containing the username that created the RMA request.
1426.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1427.      */
1428.     public String getRMACreatedBy(String rmaId) throws SQLException {
1429.         // Set up the SQL query.
1430.         query = "select createdBy from rmaDetails where rmaId =  '" + rmaId + "';";
1431.
1432.         // Fetch the results.
1433.         connect();
1434.         executeQuery(query);
1435.
1436.         String result = "";
1437.         if (rs.next())
1438.             result = rs.getString("createdBy");
1439.
1440.         // Clean up the connection.
1441.         closeConnection();
1442.
1443.         // Return the result.
1444.         return result;
1445.     }
1446.
1447.     /** getRMAProduct returns a PurchaseOrderProduct containing the details of the product
    that is being returned
1448.      *  in the given RMA.
1449.      * @param rmaId The {@link String} ID of the RMA whose created username we wish to
    fetch.
1450.      * @return A PurchaseOrderProduct containing the purchase order product information of
    the item being returned
1451.      * in the requested RMA.
1452.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1453.      */
1454.     public PurchaseOrderProduct getRMAProduct(String rmaId) throws SQLException {
1455.         // Set up the SQL query.
1456.         query = "select pop.purchaseOrderProductId, pop.poNumber, p.productName,
    pc.categoryName, pop.quantity, " +
1457.                 "pop.orderDate, pop.deliverDate " +
1458.                 "from rmaDetails rd, purchaseOrderProducts pop, products p,
    productCategories pc " +
1459.                 "where rd.rmaId =  '" + rmaId + "' " +
1460.                 "and rd.purchaseOrderProductId = pop.purchaseOrderProductId " +
1461.                 "and pop.categoryId = p.categoryId " +
1462.                 "and pop.productId = p.productId " +
1463.                 "and p.categoryId = pc.categoryId;";
1464.
1465.         // Fetch the results.
1466.         connect();
1467.         executeQuery(query);
1468.
1469.         PurchaseOrderProduct result = null;
1470.         if (rs.next())
1471.             result = new PurchaseOrderProduct(
1472.                 rs.getInt("purchaseOrderProductId"),
1473.                 rs.getString("poNumber"),
1474.                 rs.getString("productName"),
1475.                 rs.getString("categoryName"),
1476.                 rs.getInt("quantity"),
1477.                 rs.getObject("orderDate", LocalDate.class),
1478.                 rs.getObject("deliverDate", LocalDate.class)
```

```
1479.             );
1480.
1481.         // Clean up the connection.
1482.         closeConnection();
1483.
1484.         // Return the result.
1485.         return result;
1486.     }
1487.
1488.     /** updateRMAProduct updates the product being returned in the RMA with the passed-in
    int id.
1489.      * @param rmaId The {@link String} ID of the RMA to update.
1490.      * @param purchaseOrderProductId The new unique int value to set for the new product.
1491.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1492.      */
1493.     public void updateRMAProduct(String rmaId, int purchaseOrderProductId) throws
    SQLException {
1494.         // Setup the SQL query to update the database.
1495.         query = "update rmaDetails " +
1496.                 "set purchaseOrderProductId = ? " +
1497.                 "where rmaId = ?;";
1498.
1499.         // Connect to the database and execute the query.
1500.         connect();
1501.         createPreparedStatement(query);
1502.         pStmt.setString(2, rmaId);
1503.         pStmt.setInt(1, purchaseOrderProductId);
1504.         pStmt.executeUpdate();
1505.         closeConnection();
1506.     }
1507.
1508.     /** getRMAReturnQuantity fetches the int number of products being returned by the
    customer for the given {@link String}
1509.      *  RMA ID.
1510.      * @param rmaId The {@link String} ID of the RMA whose return quantity we wish to
    fetch.
1511.      * @return An int containing the return quantity for the item in the given RMA.
    Returns a negative value if the
1512.      * return quantity is not found.
1513.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1514.      */
1515.     public int getRMAReturnQuantity(String rmaId) throws SQLException {
1516.         // Set up the SQL query.
1517.         query = "select returnQuantity from rmaDetails where rmaId =  '" + rmaId + "';";
1518.
1519.         // Fetch the results.
1520.         connect();
1521.         executeQuery(query);
1522.
1523.         int result = -1;
1524.         if (rs.next())
1525.             result = rs.getInt("returnQuantity");
1526.
1527.         // Clean up the connection.
1528.         closeConnection();
1529.
1530.         // Return the result.
1531.         return result;
1532.     }
1533.
1534.     /** updateRMAReturnQuantity updates the int value for the amount of product being
    returned by the customer
1535.      *  for the given {@link String} RMA ID.
```

```
1536.      * @param rmaId The {@link String} ID of the RMA to update.
1537.      * @param returnQuantity The new int value to set for the product in the RMA.
1538.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1539.      */
1540.     public void updateRMAReturnQuantity(String rmaId, int returnQuantity) throws
    SQLException {
1541.         // Setup the SQL query to update the database.
1542.         query = "update rmaDetails " +
1543.                 "set returnQuantity = ? " +
1544.                 "where rmaId = ?;";
1545.
1546.         // Connect to the database and execute the query.
1547.         connect();
1548.         createPreparedStatement(query);
1549.         pStmt.setString(2, rmaId);
1550.         pStmt.setInt(1, returnQuantity);
1551.         pStmt.executeUpdate();
1552.         closeConnection();
1553.     }
1554.
1555.     /** getRMAReturnLabelTracker fetches the {@link String} return label tracking number
    stored in the referenced RMA.
1556.      * @param rmaId The {@link String} ID of the RMA whose return label tracking number we
    wish to fetch.
1557.      * @return A {@link String} containing the return label tracking number for the item
    in the given RMA.
1558.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1559.      */
1560.     public String getRMAReturnLabelTracker(String rmaId) throws SQLException {
1561.         // Set up the SQL query.
1562.         query = "select returnLabelTracker from rmaDetails where rmaId =  '" + rmaId +
    "';";
1563.
1564.         // Fetch the results.
1565.         connect();
1566.         executeQuery(query);
1567.
1568.         String result = "";
1569.         if (rs.next())
1570.             result = rs.getString("returnLabelTracker");
1571.
1572.         // Clean up the connection.
1573.         closeConnection();
1574.
1575.         // Return the result.
1576.         return result;
1577.     }
1578.
1579.     /** updateRMAReturnLabelTracker updates the requested RMA's details with the new
    passed-in {@link String} return label tracking ID.
1580.      * @param rmaId The {@link String} ID of the RMA to update.
1581.      * @param returnLabelTracker The new tracking ID to assign to the RMA.
1582.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1583.      */
1584.     public void updateRMAReturnLabelTracker(String rmaId, String returnLabelTracker)
    throws SQLException {
1585.         // Setup the SQL query to update the database.
1586.         query = "update rmaDetails " +
1587.                 "set returnLabelTracker = ? " +
1588.                 "where rmaId = ?;";
1589.
1590.         // Connect to the database and execute the query.
```

```
1591.         connect();
1592.         createPreparedStatement(query);
1593.         pStmt.setString(2, rmaId);
1594.         pStmt.setString(1, returnLabelTracker);
1595.         pStmt.executeUpdate();
1596.         closeConnection();
1597.     }
1598.
1599.     /** getRMAInitialEvaluation fetches the {@link String} initial evaluation done by an
    Analyst for the requested RMA.
1600.      * @param rmaId The {@link String} ID of the RMA whose initial evaluation we wish to
    fetch.
1601.      * @return A {@link String} containing the initial evaluation of the returned product
    for the given RMA.
1602.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1603.      */
1604.     public String getRMAInitialEvaluation(String rmaId) throws SQLException {
1605.         // Set up the SQL query.
1606.         query = "select initialEvaluation from rmaDetails where rmaId =  '" + rmaId +
    "';";
1607.
1608.         // Fetch the results.
1609.         connect();
1610.         executeQuery(query);
1611.
1612.         String result = "";
1613.         if (rs.next())
1614.             result = rs.getString("initialEvaluation");
1615.
1616.         // Clean up the connection.
1617.         closeConnection();
1618.
1619.         // Return the result.
1620.         return result;
1621.     }
1622.
1623.     /** updateRMAInitialEvaluation updates the requested RMA's initial evaluation from an
    Analyst with the new
1624.      *  {@link String} passed-in initial evaluation text.
1625.      * @param rmaId The {@link String} ID of the RMA to update.
1626.      * @param initialEvaluation The updated {@link String} Initial Evaluation value to
    store in the database.
1627.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1628.      */
1629.     public void updateRMAInitialEvaluation(String rmaId, String initialEvaluation) throws
    SQLException {
1630.         // Setup the SQL query to update the database.
1631.         query = "update rmaDetails " +
1632.                 "set initialEvaluation = ? " +
1633.                 "where rmaId = ?;";
1634.
1635.         // Connect to the database and execute the query.
1636.         connect();
1637.         createPreparedStatement(query);
1638.         pStmt.setString(2, rmaId);
1639.         pStmt.setString(1, initialEvaluation);
1640.         pStmt.executeUpdate();
1641.         closeConnection();
1642.     }
1643.
1644.     /** getRMAEngineeringEvaluation fetches the {@link String} engineering evaluation done
    by an Engineer for the requested RMA.
```

```
1645.      * @param rmaId The {@link String} ID of the RMA whose engineering evaluation we wish
    to fetch.
1646.      * @return A {@link String} containing the engineering evaluation of the returned
    product for the given RMA.
1647.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1648.      */
1649.     public String getRMAEngineeringEvaluation(String rmaId) throws SQLException {
1650.         // Set up the SQL query.
1651.         query = "select engineeringEvaluation from rmaDetails where rmaId =  '" + rmaId +
    "';";
1652.
1653.         // Fetch the results.
1654.         connect();
1655.         executeQuery(query);
1656.
1657.         String result = "";
1658.         if (rs.next())
1659.             result = rs.getString("engineeringEvaluation");
1660.
1661.         // Clean up the connection.
1662.         closeConnection();
1663.
1664.         // Return the result.
1665.         return result;
1666.     }
1667.
1668.     /** updateRMAEngineeringEvaluation updates the requested RMA's engineering evaluation
    from an Engineer with the new {@link String}
1669.      *  passed-in engineering evaluation text.
1670.      * @param rmaId The {@link String} ID of the RMA to update.
1671.      * @param engineeringEvaluation The new {@link String} engineering evaluation to save
    in the database.
1672.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1673.      */
1674.     public void updateRMAEngineeringEvaluation(String rmaId, String engineeringEvaluation)
    throws SQLException {
1675.         // Setup the SQL query to update the database.
1676.         query = "update rmaDetails " +
1677.                 "set engineeringEvaluation = ? " +
1678.                 "where rmaId = ?;";
1679.
1680.         // Connect to the database and execute the query.
1681.         connect();
1682.         createPreparedStatement(query);
1683.         pStmt.setString(2, rmaId);
1684.         pStmt.setString(1, engineeringEvaluation);
1685.         pStmt.executeUpdate();
1686.         closeConnection();
1687.     }
1688.
1689.     /** getRMADisposition fetches the {@link String} disposition for the requested RMA.
1690.      * @param rmaId The {@link String} ID of the RMA whose disposition we wish to fetch.
1691.      * @return A {@link String} containing the disposition of the returned product for the
    given RMA.
1692.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1693.      */
1694.     public String getRMADisposition(String rmaId) throws SQLException {
1695.         // Set up the SQL query.
1696.         query = "select d.disposition " +
1697.                 "from rmaDetails rd, dispositions d " +
1698.                 "where rd.rmaId =  '" + rmaId + "' " +
1699.                 "and rd.dispositionId = d.dispositionId;";
```

```
1700.
1701.         // Fetch the results.
1702.         connect();
1703.         executeQuery(query);
1704.
1705.         String result = "";
1706.         if (rs.next())
1707.             result = rs.getString("disposition");
1708.
1709.         // Clean up the connection.
1710.         closeConnection();
1711.
1712.         // Return the result.
1713.         return result;
1714.     }
1715.
1716.     /** updateRMADisposition updates the disposition ID stored in the specified RMA using
    the passed-in {@link String} disposition.
1717.      * @param rmaId The {@link String} ID of the RMA to update.
1718.      * @param disposition The {@link String} disposition description to look up and then
    use to update.
1719.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1720.      */
1721.     public void updateRMADisposition(String rmaId, String disposition) throws SQLException
    {
1722.         // Setup the SQL query to update the database.
1723.         query = "update rmaDetails " +
1724.                 "set dispositionId = (select dispositionId from dispositions where
    disposition = ?) " +
1725.                 "where rmaId = ?;";
1726.
1727.         // Connect to the database and execute the query.
1728.         connect();
1729.         createPreparedStatement(query);
1730.         pStmt.setString(2, rmaId);
1731.         pStmt.setString(1, disposition);
1732.         pStmt.executeUpdate();
1733.         closeConnection();
1734.     }
1735.
1736.     /** getRMADispositionNotes fetches the {@link String} disposition notes for the
    requested RMA.
1737.      * @param rmaId The {@link String} ID of the RMA whose disposition notes we wish to
    fetch.
1738.      * @return A {@link String} containing the disposition notes for the returned product
    in the given RMA.
1739.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1740.      */
1741.     public String getRMADispositionNotes(String rmaId) throws SQLException {
1742.         // Set up the SQL query.
1743.         query = "select dispositionNotes from rmaDetails where rmaId =  '" + rmaId + "';";
1744.
1745.         // Fetch the results.
1746.         connect();
1747.         executeQuery(query);
1748.
1749.         String result = "";
1750.         if (rs.next())
1751.             result = rs.getString("dispositionNotes");
1752.
1753.         // Clean up the connection.
1754.         closeConnection();
1755.
```

```
1756.         // Return the result.
1757.         return result;
1758.     }
1759.
1760.     /** updateRMADispositionNotes updates the disposition notes stored in the specified
    RMA using the passed-in {@link String} notes.
1761.      * @param rmaId The {@link String} ID of the RMA to update.
1762.      * @param dispositionNotes The new {@link String} dispositionNotes to save to the
    database.
1763.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1764.      */
1765.     public void updateRMADispositionNotes(String rmaId, String dispositionNotes) throws
    SQLException {
1766.         // Setup the SQL query to update the database.
1767.         query = "update rmaDetails " +
1768.                 "set dispositionNotes = ? " +
1769.                 "where rmaId = ?;";
1770.
1771.         // Connect to the database and execute the query.
1772.         connect();
1773.         createPreparedStatement(query);
1774.         pStmt.setString(2, rmaId);
1775.         pStmt.setString(1, dispositionNotes);
1776.         pStmt.executeUpdate();
1777.         closeConnection();
1778.     }
1779.
1780.     /** getRMAReplacementTrackingNumber fetches the {@link String} tracking number that
    will be used to ship the replacement
1781.      *  back to the customer in the requested RMA.
1782.      * @param rmaId The {@link String} ID of the RMA whose replacement tracking number we
    wish to fetch.
1783.      * @return A {@link String} containing the replacement tracking number for the
    replacement or repair for the given RMA.
1784.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1785.      */
1786.     public String getRMAReplacementTrackingNumber(String rmaId) throws SQLException {
1787.         // Set up the SQL query.
1788.         query = "select replacementTrackingNumber from rmaDetails where rmaId =  '" +
    rmaId + "';";
1789.
1790.         // Fetch the results.
1791.         connect();
1792.         executeQuery(query);
1793.
1794.         String result = "";
1795.         if (rs.next())
1796.             result = rs.getString("replacementTrackingNumber");
1797.
1798.         // Clean up the connection.
1799.         closeConnection();
1800.
1801.         // Return the result.
1802.         return result;
1803.     }
1804.
1805.     /** updateRMAReplacementTrackingNumber updates the specified RMA's tracking number
    used to ship the replacement product.
1806.      * @param rmaId The {@link String} ID of the RMA to update.
1807.      * @param replacementTrackingNumber The {@link String} new replacement tracking number
    to store in the database.
1808.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
```

```
1809.      */
1810.     public void updateRMAReplacementTrackingNumber(String rmaId, String
    replacementTrackingNumber) throws SQLException {
1811.         // Setup the SQL query to update the database.
1812.         query = "update rmaDetails " +
1813.                 "set replacementTrackingNumber = ? " +
1814.                 "where rmaId = ?;";
1815.
1816.         // Connect to the database and execute the query.
1817.         connect();
1818.         createPreparedStatement(query);
1819.         pStmt.setString(2, rmaId);
1820.         pStmt.setString(1, replacementTrackingNumber);
1821.         pStmt.executeUpdate();
1822.         closeConnection();
1823.     }
1824.
1825.     /** getRMAReplacementShipDate fetches the {@link LocalDate} of the date that the
    replacement will be shipped to the customer
1826.      *  for the given RMA.
1827.      * @param rmaId The {@link String} ID of the RMA whose replacement ship date we wish
    to fetch.
1828.      * @return A {@link LocalDate} containing the replacement ship date of the replacement
    or repair for the given RMA.
1829.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1830.      */
1831.     public LocalDate getRMAReplacementShipDate(String rmaId) throws SQLException {
1832.         // Set up the SQL query.
1833.         query = "select replacementShipDate from rmaDetails where rmaId =  '" + rmaId +
    "';";
1834.
1835.         // Fetch the results.
1836.         connect();
1837.         executeQuery(query);
1838.
1839.         LocalDate result = null;
1840.         if (rs.next())
1841.             result = rs.getObject("replacementShipDate", LocalDate.class);
1842.
1843.         // Clean up the connection.
1844.         closeConnection();
1845.
1846.         // Return the result.
1847.         return result;
1848.     }
1849.
1850.     /** updateRMAReplacementShipDate updates the {@link LocalDate} date with the passed-in
    date the replacement will be shipped to
1851.      *  the customer in the RMA with the specified {@link String} ID.
1852.      * @param rmaId The {@link String} ID of the RMA to update.
1853.      * @param replacementShipDate The new {@link LocalDate} date to store in the database.
1854.      * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1855.      */
1856.     public void updateRMAReplacementShipDate(String rmaId, LocalDate replacementShipDate)
    throws SQLException {
1857.         // Setup the SQL query to update the database.
1858.         query = "update rmaDetails " +
1859.                 "set replacementShipDate = ? " +
1860.                 "where rmaId = ?;";
1861.
1862.         // Connect to the database and execute the query.
1863.         connect();
1864.         createPreparedStatement(query);
```

```
1865.          pStmt.setString(2, rmaId);
1866.          pStmt.setObject(1, replacementShipDate);
1867.          pStmt.executeUpdate();
1868.          closeConnection();
1869.      }
1870.
1871.      /** getRMAShipReplacementRepair fetches the boolean value of whether we will be
    shipping a replacement or repair to
1872.       *  the customer referenced in the RMA with the given {@link String} ID.
1873.       * @param rmaId The {@link String} ID of the RMA to update.
1874.       * @return A boolean containing the shipReplacementRepair value.
1875.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1876.       */
1877.      public boolean getRMAShipReplacementRepair(String rmaId) throws SQLException {
1878.          // Set up the SQL query.
1879.          query = "select shipReplacementRepair from rmaDetails where rmaId =  '" + rmaId +
    "';";
1880.
1881.          // Fetch the results.
1882.          connect();
1883.          executeQuery(query);
1884.
1885.          boolean result = false;
1886.          if (rs.next())
1887.              result = rs.getBoolean("shipReplacementRepair");
1888.
1889.          // Clean up the connection.
1890.          closeConnection();
1891.
1892.          // Return the result.
1893.          return result;
1894.      }
1895.
1896.      /** updateRMAShipReplacementRepair updates the RMA with the given {@link String} ID
    with the new shipReplacementRepair
1897.       *  boolean value.
1898.       * @param rmaId The {@link String} ID of the RMA to update.
1899.       * @param shipReplacementRepair The new boolean value to store in the database.
1900.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1901.       */
1902.      public void updateRMAShipReplacementRepair(String rmaId, boolean
    shipReplacementRepair) throws SQLException {
1903.          // Setup the SQL query to update the database.
1904.          query = "update rmaDetails " +
1905.                  "set shipReplacementRepair = ? " +
1906.                  "where rmaId = ?;";
1907.
1908.          // Connect to the database and execute the query.
1909.          connect();
1910.          createPreparedStatement(query);
1911.          pStmt.setString(2, rmaId);
1912.          pStmt.setBoolean(1, shipReplacementRepair);
1913.          pStmt.executeUpdate();
1914.          closeConnection();
1915.      }
1916.
1917.      /** deleteRMA deletes the RMA with the given {@link String} ID.
1918.       * @param rmaId The {@link String} ID of the RMA to delete.
1919.       * @throws SQLException If there is an issue connecting to the database or an issue
    with the SQL query.
1920.       */
1921.      public void deleteRMA(String rmaId) throws SQLException {
1922.          // Setup the SQL query to update the database.
```

```
1923.         query = "delete from rma where rmaId = ?;";
1924.
1925.         // Connect to the database and execute the query.
1926.         connect();
1927.         createPreparedStatement(query);
1928.         pStmt.setString(1, rmaId);
1929.         pStmt.executeUpdate();
1930.         closeConnection();
1931.     }
1932. }
1933.
```

LoginController.java

```
1.  package RMA;
2.
3.  import javafx.beans.value.ChangeListener;
4.  import javafx.beans.value.ObservableValue;
5.  import javafx.event.ActionEvent;
6.  import javafx.event.Event;
7.  import javafx.event.EventHandler;
8.  import javafx.fxml.FXML;
9.  import javafx.fxml.FXMLLoader;
10. import javafx.fxml.Initializable;
11. import javafx.scene.Node;
12. import javafx.scene.Parent;
13. import javafx.scene.Scene;
14. import javafx.scene.control.*;
15. import javafx.scene.layout.Region;
16. import javafx.stage.Stage;
17. import javafx.stage.WindowEvent;
18.
19. import java.io.IOException;
20. import java.net.URL;
21. import java.sql.SQLException;
22. import java.util.Optional;
23. import java.util.ResourceBundle;
24.
25. /** LoginController contains the logic and field references to the GUI controls
26.  *  on the Login form.
27.  */
28. public class LoginController implements Initializable {
29.     /** txtUsername references the TextField where the user enters their username to
30.      *  log into the RMA database.
31.      */
32.     @FXML
33.     private TextField txtUsername;
34.     /** txtPassword references the TextField where the user enters their password to
35.      *  log into the RMA database.
36.       */
37.     @FXML
38.     private TextField txtPassword;
39.     /** chobInstances contains the list of SQL Server instances found on the network.
40.      */
41.     @FXML
42.     private ChoiceBox<String> chobInstances;
43.     /** btnConnect is the button the user presses to connect to the selected instance.
44.      */
45.     @FXML
```

```
46.     private Button btnConnect;
47.     /** model is the data model for the Login form.
48.      */
49.     private LoginModel model;
50.
51.     /** Default empty constructor, used when {@link javafx.fxml.FXMLLoader} loads the fxml
    file.
52.      */
53.     public LoginController() {}
54.
55.     /** initialize ties together the GUI's elements with the LoginModel model and creates
    any
56.      *  necessary {@link ChangeListener}s on the controls.
57.      *  (Requirement 1.1.1, 1.2.1, 1.4.1)
58.      * @param url The {@link URL} location of the fxml file that describes the Login form's
    layout.
59.      * @param rb {@link ResourceBundle} that contains the data for the user's locale.
60.      */
61.     @Override
62.     public void initialize(URL url, ResourceBundle rb) {
63.         model = new LoginModel();
64.         txtUsername.textProperty().bindBidirectional(model.userProperty());
65.         txtUsername.textProperty().addListener(
66.             new ChangeListener<String>() {
67.                 /** changed listens for changes to the Username {@link TextField} to decide
    whether the Connect {@link Button}
68.                  *  should be enabled.
69.                  * @param observableValue The {@link ObservableValue} that changed.
70.                  * @param oldValue The {@link String} previous value.
71.                  * @param newValue The {@link String} new value.
72.                  */
73.                 @Override
74.                 public void changed(ObservableValue<? extends String> observableValue,
    String oldValue, String newValue) {
75.                     if (chobInstances.getSelectionModel().selectedItemProperty().getValue()
    != null)
76.                         if (!txtPassword.textProperty().getValue().isEmpty() &&
    !chobInstances.getSelectionModel().selectedItemProperty().getValue().isEmpty())
77.                             btnConnect.setDisable(newValue.isEmpty() || newValue.isBlank());
78.                         else
79.                             btnConnect.setDisable(true);
80.                 }
81.             }
82.         );
83.         txtPassword.textProperty().bindBidirectional(model.passProperty());
84.         txtPassword.textProperty().addListener(
85.             new ChangeListener<String>() {
86.                 /** changed listens for changes to the Password {@link TextField} to decide
    whether the Connect {@link Button}
87.                  *  should be enabled.
88.                  * @param observableValue The {@link ObservableValue} that changed.
89.                  * @param oldValue The {@link String} previous value.
90.                  * @param newValue The {@link String} new value.
91.                  */
92.                 @Override
93.                 public void changed(ObservableValue<? extends String> observableValue,
    String oldValue, String newValue) {
94.                     if (chobInstances.getSelectionModel().selectedItemProperty().getValue()
    != null)
95.                         if (!txtUsername.textProperty().getValue().isEmpty() &&
    !chobInstances.getSelectionModel().selectedItemProperty().getValue().isEmpty())
96.                             btnConnect.setDisable(newValue.isEmpty() || newValue.isBlank());
97.                         else
98.                             btnConnect.setDisable(true);
99.                 }
```

```
100.             }
101.         );
102.         chobInstances.setItems(model.instancesProperty());
103.         chobInstances.valueProperty().bindBidirectional(model.selectedInstanceProperty());
104.         chobInstances.getSelectionModel().selectedItemProperty().addListener(
105.             new ChangeListener<String>() {
106.                 /** changed listens for changes to the Instances {@link ChoiceBox} in
    order to decide whether to enable
107.                  *  the Connect {@link Button}.
108.                  * @param observableValue The {@link ObservableValue} that changed.
109.                  * @param oldValue The {@link String} previous value.
110.                  * @param newValue The {@link String} new value.
111.                  */
112.                 @Override
113.                 public void changed(ObservableValue<? extends String> observableValue,
    String oldValue, String newValue) {
114.                     btnConnect.setDisable(newValue.isEmpty() || newValue.isBlank());
115.                 }
116.             }
117.         );
118.     }
119.
120.     /** chobInstancesOnAction contains the logic to process the user clicking on the
    chobInstances {@link ChoiceBox}.
121.      *  (Requirement 1.4.2)
122.      * @param event The {@link Event} information pertaining to the user's action.
123.      */
124.     @FXML
125.     private void chobInstancesOnAction(Event event) {
126.         if (txtUsername.getText() == null) {
127.             Alert a = new Alert(Alert.AlertType.ERROR);
128.             a.setContentText("Please fill in your username before selecting an
    instance!");
129.             a.show();
130.             txtUsername.requestFocus();
131.             event.consume();
132.         } else if (txtUsername.getText().isEmpty()) {
133.             Alert a = new Alert(Alert.AlertType.ERROR);
134.             a.setContentText("Please fill in your username before selecting an
    instance!");
135.             a.show();
136.             txtUsername.requestFocus();
137.             event.consume();
138.         } else if (txtPassword.getText() == null) {
139.             Alert a = new Alert(Alert.AlertType.ERROR);
140.             a.setContentText("Please fill in your password before selecting an
    instance!");
141.             a.show();
142.             txtPassword.requestFocus();
143.             event.consume();
144.         } else if (txtPassword.getText().isEmpty()) {
145.             Alert a = new Alert(Alert.AlertType.ERROR);
146.             a.setContentText("Please fill in your password before selecting an
    instance!");
147.             a.show();
148.             txtPassword.requestFocus();
149.             event.consume();
150.         }
151.     }
152.
153.     /** btnConnectOnAction contains the logic to process the user clicking on the
154.      *  Connect {@link Button}. In particular, it first checks that the user has filled
155.      *  out every field, and then attempts to connect to the RMA database stored in the
156.      *  user-selected SQL Server instance. If successful, it initializes an instance of
157.      *  DBService, and then launches the main RMAListView window. If it is unable to
```

```
158.        *  connect or find an RMA database in the given instance, a pop-up is shown.
159.        * @param event The {@link ActionEvent} information pertaining to the user's action.
160.        */
161.       @FXML
162.       private void btnConnectOnAction(ActionEvent event) {
163.           btnConnect.setDisable(true);
164.           if (!model.getUser().isEmpty() &&
165.               !model.getUser().isBlank() &&
166.               !model.getPass().isEmpty() &&
167.               !model.getPass().isBlank() &&
168.               !model.getSelectedInstance().isEmpty() &&
169.               !model.getSelectedInstance().isBlank()
170.           )
171.               try {
172.                   // Initialize DBService.
173.                   DBService.getInstance(model.getUser(), model.getPass(),
    model.getSelectedInstance());
174.
175.                   // Load the RMAListView screen.
176.                   FXMLLoader loader = new FXMLLoader();
177.                   loader.setLocation(getClass().getResource("RMAListView_V4.fxml"));
178.                   Parent root = loader.load();
179.                   Scene scene = new Scene(root);
180.                   Stage stage = new Stage();
181.                   stage.setTitle("RMA List View");
182.                   stage.setScene(scene);
183.                   stage.getScene().getWindow().setOnCloseRequest(
184.                           new EventHandler<WindowEvent>() {
185.                               /** handle listens for the WINDOW_CLOSE_REQUEST in order to
    ask the user whether they
186.                                *  want to close before continuing.
187.                                * @param windowEvent The {@link WindowEvent} that was passed
    to this method.
188.                                */
189.                               @Override
190.                               public void handle(WindowEvent windowEvent) {
191.                                   if
    (windowEvent.getEventType().equals(WindowEvent.WINDOW_CLOSE_REQUEST)) {
192.                                       // Ask the user if they want to close.
193.                                       ButtonType confirm = new ButtonType("Confirm",
    ButtonBar.ButtonData.OK_DONE); // Change the button text from OK to Confirm
194.                                       Alert a = new Alert(
195.                                               Alert.AlertType.CONFIRMATION,
196.                                               "Are you sure you want to exit?",
197.                                               ButtonType.NO,
198.                                               confirm
199.                                       );
200.                                       a.setTitle("RMA List View Close Confirmation");
201.                                       a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
202.                                       Optional<ButtonType> result = a.showAndWait();
203.
204.                                       if (result.isPresent() && result.get() != confirm)
205.                                           windowEvent.consume();
206.                                   }
207.                               }
208.                           }
209.                   );
210.
211.                   stage.show();
212.
213.                   // Close this window.
214.                   ((Node) event.getSource()).getScene().getWindow().hide();
215.               } catch (SQLException e) {
216.                   Alert a = new Alert(Alert.AlertType.ERROR);
```

```
217.                     a.setContentText("Error while connecting to SQL Server database!" +
    System.lineSeparator() + e.getLocalizedMessage());
218.                     a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
219.                     a.show();
220.                 } catch (IOException e) {
221.                     Alert a = new Alert(Alert.AlertType.ERROR);
222.                     a.setContentText("Error while initializing form!" + System.lineSeparator()
    + e);
223.                     e.printStackTrace();
224.                     a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
225.                     a.show();
226.                 }
227.             // Re-enable the button.
228.             btnConnect.setDisable(false);
229.         }
230.     }
231.
```

RMAListViewController.java

```
1.  package RMA;
2.
3.  import javafx.beans.property.SimpleStringProperty;
4.  import javafx.beans.property.StringProperty;
5.  import javafx.beans.value.ChangeListener;
6.  import javafx.beans.value.ObservableValue;
7.  import javafx.collections.FXCollections;
8.  import javafx.collections.ObservableList;
9.  import javafx.collections.transformation.FilteredList;
10. import javafx.event.ActionEvent;
11. import javafx.event.EventHandler;
12. import javafx.fxml.FXML;
13. import javafx.fxml.FXMLLoader;
14. import javafx.fxml.Initializable;
15. import javafx.scene.Parent;
16. import javafx.scene.Scene;
17. import javafx.scene.control.*;
18. import javafx.scene.control.cell.CheckBoxTableCell;
19. import javafx.scene.control.cell.PropertyValueFactory;
20. import javafx.scene.input.MouseEvent;
21. import javafx.scene.layout.Region;
22. import javafx.stage.Modality;
23. import javafx.stage.Stage;
24. import javafx.stage.WindowEvent;
25. import javafx.util.Callback;
26.
27. import java.io.IOException;
28. import java.net.URL;
29. import java.sql.SQLException;
30. import java.time.LocalDateTime;
31. import java.time.format.DateTimeFormatter;
32. import java.util.ArrayList;
33. import java.util.Optional;
34. import java.util.ResourceBundle;
35.
36. /** RMAListViewController contains the logic and field references to the GUI controls on the
    RMAListView form.
37.  */
38. public class RMAListViewController implements Initializable {
39.     /** lblItemCount contains the number of items in the RMA list and the last time the list
    was updated.
40.      *  (Requirement 2.1)
41.      */
42.     @FXML
43.     private Label lblItemCount;
44.     /** txtSearch is a {@link TextField} used to filter the RMA list by an RMA ID.
45.      *  (Requirement 2.3)
46.      */
47.     @FXML
48.     private TextField txtSearch;
49.     /** btnNewRMA brings up a new RMAForm for creating a new RMA.
50.      *  (Requirement 2.4)
51.      */
52.     @FXML
53.     private Button btnNewRMA;
54.     /** btnRefresh is used to fetch the latest RMA list from the SQL Server RMA database.
55.      *  (Requirement 2.5)
56.      */
57.     @FXML
58.     private Button btnRefresh;
59.     /** btnDelete deletes the selected RMAs in the tblRMASummary list.
```

```
60.      *  (Requirement 2.7)
61.      */
62.     @FXML
63.     private Button btnDelete;
64.     /** tblRMASummary contains the list of open RMA requests available to this user's role.
65.      *  (Requirement 2.2, 2.2.1)
66.      */
67.     @FXML
68.     private TableView<RMAListViewModel> tblRMASummary;
69.     /** tblList is the {@link ObservableList} representing the contents of tblRMASummary.
70.      */
71.     private ObservableList<RMAListViewModel> tblList;
72.     /** filteredList is used to filter tblList when the user searches for an RMA ID in
    txtSearch.
73.      *  (Requirement 2.3.1)
74.      */
75.     private FilteredList<RMAListViewModel> filteredList;
76.     /** colShouldDelete is used to mark whether this RMA should be deleted from the
    database.
77.      *  (Requirement 2.7, 2.2.1.1)
78.      */
79.     @FXML
80.     private TableColumn<RMAListViewModel, Boolean> colShouldDelete;
81.     /** colCustomerName contains the customer name associated with the RMA request.
82.      *  (Requirement 2.2.1.1)
83.      */
84.     @FXML
85.     private TableColumn<RMAListViewModel, String> colCustomerName;
86.     /** colBusinessName contains the business name associated with the RMA request.
87.      *  (Requirement 2.2.1.1)
88.      */
89.     @FXML
90.     private TableColumn<RMAListViewModel, CustomerAddress> colBusinessName;
91.     /** colRMAId contains the RMA's ID.
92.      *  (Requirement 2.2.1.1)
93.      */
94.     @FXML
95.     private TableColumn<RMAListViewModel, String> colRMAId;
96.     /** colRMAStatus contains the RMA's status.
97.      *  (Requirement 2.2.1.1)
98.      */
99.     @FXML
100.      private TableColumn<RMAListViewModel, String> colRMAStatus;
101.      /** colProduct contains the name of the product being returned.
102.       *  (Requirement 2.2.1.1)
103.       */
104.      @FXML
105.      private TableColumn<RMAListViewModel, PurchaseOrderProduct> colProduct;
106.      /** colReplaceRepairIndicator is used to show whether the customer's
107.       *  replacement or repair has been sent.
108.       *  (Requirement 2.2.1.1)
109.       */
110.      @FXML
111.      private TableColumn<RMAListViewModel, Boolean> colReplaceRepairIndicator;
112.      /** colCustomerInfoShipAddress displays information about the customer and
113.       *  their shipping address.
114.       *  (Requirement 2.2.1.1)
115.       */
116.      @FXML
117.      private TableColumn<RMAListViewModel, CustomerAddress> colCustomerInfoShipAddress;
118.      /** colReturnQuantity displays the quantity of product that the customer is
119.       *  returning.
120.       *  (Requirement 2.2.1.1)
121.       */
122.      @FXML
```

```
123.      private TableColumn<RMAListViewModel, Integer> colReturnQuantity;
124.      /** lastUpdated keeps track of the last time the RMA list was updated.
125.       *  (Requirement 2.1)
126.       */
127.      private LocalDateTime lastUpdated;
128.      /** tableCount manages the text above the TableView listing the number of items and
    the last update date and time
129.       *  (Requirement 2.1)
130.       */
131.      private StringProperty tableCount;
132.
133.
134.      /** Default empty constructor, used when {@link javafx.fxml.FXMLLoader} loads the fxml
    file.
135.       */
136.      public RMAListViewController() {}
137.
138.      /** initialize ties together the GUI's elements with the RMAListViewModel model and
139.       *  creates any necessary {@link ChangeListener}s on the controls.
140.       *  (Requirement 2.2.2, 2.4.2, 2.6, 2.7.1, 5.2.8.2)
141.       * @param url The {@link URL} location of the fxml file that describes
142.       *            the RMAListView window elements and their layout.
143.       * @param rb {@link ResourceBundle} that contains locale-specific data.
144.       */
145.      @Override
146.      public void initialize(URL url, ResourceBundle rb) {
147.          // First set up the ObservableList and FilteredList.
148.          tblList = FXCollections.observableArrayList();
149.          filteredList = new FilteredList<RMAListViewModel>(tblList);
150.          tblRMASummary.setItems(filteredList);
151.          tblRMASummary.setEditable(true);
152.          lastUpdated = LocalDateTime.now();
153.          tableCount = new SimpleStringProperty();
154.          lblItemCount.textProperty().bindBidirectional(tableCount);
155.
156.          // Add ChangeListener to txtSearch to update what the RMAListViewModels
    FilteredList shows whenever its contents change.
157.          txtSearch.textProperty().addListener(
158.              new ChangeListener<String>() {
159.                  /** changed listens for changes to the input of the txtSearch {@link
    TextField} in order to filter the
160.                   *  {@link TableView} list.
161.                   * @param observableValue The {@link ObservableValue} that changed.
162.                   * @param oldValue The {@link String} previous value.
163.                   * @param newValue The {@link String} new value.
164.                   */
165.                  @Override
166.                  public void changed(ObservableValue<? extends String> observableValue,
    String oldValue, String newValue) {
167.                      // Update the TableView list filter.
168.                      filteredList.setPredicate(model -> {
169.                          if (newValue == null)
170.                              return true;
171.                          else if (newValue.isEmpty())
172.                              return true;
173.                          else return model.getRMAId().startsWith(newValue);
174.                      });
175.
176.                      // Update the item count.
177.                      tableCount.setValue(filteredList.size() + " items in list. Last
    updated: " + lastUpdated.format(DateTimeFormatter.ofPattern("MM/dd/yyyy HH:mm:ss")));
178.                  }
179.              }
180.          );
181.
```

```
182.          // Set CellValueFactory for each column.
183.          colShouldDelete.setCellValueFactory(new PropertyValueFactory<RMAListViewModel,
    Boolean>("shouldDelete"));
184.          colShouldDelete.setCellFactory(
185.              new
    Callback<TableColumn<RMAListViewModel,Boolean>,TableCell<RMAListViewModel,Boolean>>(){
186.                  /** call handles the callback from input being sent to the cells in the
    colShouldDelete {@link TableColumn}.
187.                   * @param p The {@link TableColumn} referenced by the call.
188.                   * @return The format of the {@link TableCell}.
189.                   */
190.                  @Override
191.                  public TableCell<RMAListViewModel,Boolean>
    call(TableColumn<RMAListViewModel,Boolean> p){
192.                      return new CheckBoxTableCell<>();
193.                  }
194.              }
195.          );
196.          colCustomerName.setCellValueFactory(new PropertyValueFactory<RMAListViewModel,
    String>("customerName"));
197.          colBusinessName.setCellValueFactory(new PropertyValueFactory<RMAListViewModel,
    CustomerAddress>("businessName"));
198.          colRMAId.setCellValueFactory(new PropertyValueFactory<RMAListViewModel,
    String>("rmaId"));
199.          colRMAStatus.setCellValueFactory(new PropertyValueFactory<RMAListViewModel,
    String>("status"));
200.          colProduct.setCellValueFactory(new PropertyValueFactory<RMAListViewModel,
    PurchaseOrderProduct>("product"));
201.          colReplaceRepairIndicator.setCellValueFactory(new
    PropertyValueFactory<RMAListViewModel, Boolean>("shipReplaceRepair"));
202.          colReplaceRepairIndicator.setCellFactory(
203.              new
    Callback<TableColumn<RMAListViewModel,Boolean>,TableCell<RMAListViewModel,Boolean>>(){
204.                  @Override
205.                  /** call handles the callback from input being sent to the cells in the
    colShouldDelete {@link TableColumn}.
206.                   * @param p The {@link TableColumn} referenced by the call.
207.                   * @return The format of the {@link TableCell}.
208.                   */
209.                  public TableCell<RMAListViewModel,Boolean>
    call(TableColumn<RMAListViewModel,Boolean> p){
210.                      return new CheckBoxTableCell<>();
211.                  }
212.              }
213.          );
214.          colReplaceRepairIndicator.setEditable(false);
215.          colCustomerInfoShipAddress.setCellValueFactory(new
    PropertyValueFactory<RMAListViewModel, CustomerAddress>("address"));
216.          colReturnQuantity.setCellValueFactory(new PropertyValueFactory<RMAListViewModel,
    Integer>("returnQuantity"));
217.
218.          // Add listener to TableView to open the RMA Details screen for the selected RMA.
219.          tblRMASummary.setOnMousePressed(new EventHandler<MouseEvent>() {
220.              @Override
221.              /** handle handles the user selecting entries in the {@link TableView}.
222.               * @param event The {@link MouseEvent} passed to this event handler.
223.               */
224.              public void handle(MouseEvent event) {
225.                  if (event.isPrimaryButtonDown() && event.getClickCount() == 2) {
226.                      RMAListViewModel row =
    tblRMASummary.getSelectionModel().getSelectedItem();
227.                      if (row != null) { // Open up RMA Details for the selected RMA.
228.                          try {
229.                              FXMLLoader loader = new FXMLLoader();
```

```
230.
    loader.setLocation(getClass().getResource("RMADetailScreen_V6.fxml"));
231.                                Parent root = loader.load();
232.                                // Populate the initial values for the form.
233.                                ((RMADetailsFormController)
    loader.getController()).loadRMADetails(row.getRMAId());
234.                                Scene scene = new Scene(root);
235.                                Stage stage = new Stage();
236.                                stage.initModality(Modality.APPLICATION_MODAL);
237.                                stage.setTitle("RMA Details");
238.                                stage.setScene(scene);
239.
240.                                // Make the close "X" button perform the same function as
    RMAForm.
241.                                stage.getScene().getWindow().setOnCloseRequest(
242.                                    new EventHandler<WindowEvent>() {
243.                                        /** handle listens for the WINDOW_CLOSE_REQUEST in
    order to ask the user whether they
244.                                         *  want to close before continuing.
245.                                         * @param windowEvent The {@link WindowEvent} that was
    passed to this method.
246.                                         */
247.                                        @Override
248.                                        public void handle(WindowEvent windowEvent) {
249.                                            if
    (windowEvent.getEventType().equals(WindowEvent.WINDOW_CLOSE_REQUEST)) {
250.                                                // Ask the user if they want to close.
251.                                                ButtonType confirm = new ButtonType("Confirm",
    ButtonBar.ButtonData.OK_DONE); // Change the button text from OK to Confirm
252.                                                Alert a = new Alert(
253.                                                    Alert.AlertType.CONFIRMATION,
254.                                                    "Are you sure you want to exit?",
255.                                                    confirm,
256.                                                    ButtonType.NO
257.                                                );
258.                                                a.setTitle("RMA Details Close Confirmation");
259.
    a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
260.                                                Optional<ButtonType> result = a.showAndWait();
261.
262.                                                if (result.isPresent() && result.get() !=
    confirm)
263.                                                    windowEvent.consume();
264.                                            }
265.                                        }
266.                                    }
267.                                );
268.
269.                                stage.showAndWait();
270.                                btnRefresh.fire();
271.                            } catch (IOException e) {
272.                                Alert a = new Alert(Alert.AlertType.ERROR);
273.                                a.setTitle("RMA Details Form Load Error!");
274.                                a.setContentText("Error while creating the RMA Details form!"
    + System.lineSeparator() + e);
275.                                a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
276.                                // TODO Remove once working.
277.                                e.printStackTrace();
278.                                a.showAndWait();
279.                            } catch (SQLException e) {
280.                                Alert a = new Alert(Alert.AlertType.ERROR, "Error while
    populating details in RMA Details form! Please try launching the form again!");
281.                                a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
282.                                // TODO Remove once working.
283.                                e.printStackTrace();
```

```
284.                                a.showAndWait();
285.                            }
286.                        }
287.                    }
288.                }
289.            });
290.
291.            // Fetch open RMAs appropriate for this user's role and populate tblList.
292.            getRMAList();
293.
294.            // Disable New RMA and Delete buttons if user is an Engineer.
295.            if (DBService.getInstance().getRole().equals("engineers")) {
296.                btnDelete.setDisable(true);
297.                btnNewRMA.setDisable(true);
298.            }
299.        }
300.
301.        /** getRMAList populates and refreshes the {@link ObservableList} backing the {@link
    TableView}.
302.         * (Requirement 2.5.1)
303.         */
304.        private void getRMAList() {
305.            // First clear the RMA list.
306.            tblList.clear();
307.
308.            // Fetch and loop through the list of open RMAs.
309.            try {
310.                tblList.addAll(DBService.getInstance().getRMAs());
311.            } catch (SQLException e) {
312.                Alert a = new Alert(Alert.AlertType.ERROR);
313.                a.setContentText("Error while fetching the list of open RMAs!" +
    System.lineSeparator() + e.getLocalizedMessage());
314.                a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
315.                a.showAndWait();
316.            } finally {
317.                // Update the item count.
318.                lastUpdated = LocalDateTime.now();
319.                tableCount.setValue(filteredList.size() + " items in list. Last updated: " +
    lastUpdated.format(DateTimeFormatter.ofPattern("MM/dd/yyyy HH:mm:ss")));
320.            }
321.        }
322.
323.        /** btnNewRMAOnAction opens a new RMA form.
324.         * (Requirement 2.4.1, 2.6.1, 4.17, 5.2.8.2)
325.         * @param event The {@link ActionEvent} passed to this event handler.
326.         */
327.        @FXML
328.        protected void btnNewRMAOnAction(ActionEvent event) {
329.            try {
330.                FXMLLoader loader = new FXMLLoader();
331.                loader.setLocation(getClass().getResource("RMAForm_V4.fxml"));
332.                Parent root = loader.load();
333.                // Populate the initial values for the form.
334.                ((RMAFormController) loader.getController()).getInitialRMAOptions();
335.                Scene scene = new Scene(root);
336.                Stage stage = new Stage();
337.                stage.initModality(Modality.APPLICATION_MODAL);
338.                stage.setTitle("New RMA");
339.                stage.setScene(scene);
340.
341.                // Make the close "X" button perform the same function as the Cancel button.
342.                stage.getScene().getWindow().setOnCloseRequest(
343.                    new EventHandler<WindowEvent>() {
344.                        /** handle listens for the WINDOW_CLOSE_REQUEST in order to ask the
    user whether they
```

```
345.                      *  want to close before continuing.
346.                      * @param windowEvent The {@link WindowEvent} that was passed to this
    method.
347.                      */
348.                     @Override
349.                     public void handle(WindowEvent windowEvent) {
350.                         if
    (windowEvent.getEventType().equals(WindowEvent.WINDOW_CLOSE_REQUEST)) {
351.                             // Ask the user if they want to close.
352.                             ButtonType confirm = new ButtonType("Confirm",
    ButtonBar.ButtonData.OK_DONE); // Change the button text from OK to Confirm
353.                             Alert a = new Alert(
354.                                 Alert.AlertType.CONFIRMATION,
355.                                  "Are you sure you want to exit?",
356.                                 ButtonType.NO,
357.                                 confirm
358.                             );
359.                             a.setTitle("New RMA Close Confirmation");
360.                             a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
361.                             Optional<ButtonType> result = a.showAndWait();
362.
363.                             if (result.isPresent() && result.get() != confirm)
364.                                 windowEvent.consume();
365.                         }
366.                     }
367.                 }
368.             );
369.
370.             stage.showAndWait();
371.             btnRefresh.fire();
372.         } catch (IOException e) {
373.             Alert a = new Alert(Alert.AlertType.ERROR);
374.             a.setTitle("New RMA Form Load Error!");
375.             a.setContentText("Error while creating the new RMA form!" +
    System.lineSeparator() + e);
376.             a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
377.             a.showAndWait();
378.         } catch (SQLException e) {
379.             Alert a = new Alert(Alert.AlertType.ERROR, "Error while populating initial
    values in RMA form! Please try launching the form again!");
380.             a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
381.             a.showAndWait();
382.         }
383.     }
384.
385.     /** btnRefreshOnAction handles refreshing the list of open RMAs available to this
    user's role.
386.      * (Requirement 2.5.1)
387.      * @param event The {@link ActionEvent} passed to this event handler.
388.      */
389.     @FXML
390.     protected void btnRefreshOnAction(ActionEvent event) {getRMAList();}
391.
392.     /** btnDeleteOnAction goes through the list of selected RMAs for deletion and calls
    their delete method.
393.      * (Requirement 2.7, 2.7.2, 2.7.3, 2.7.4)
394.      * @param event The {@link ActionEvent} passed to this event handler.
395.      */
396.     @FXML
397.     protected void btnDeleteOnAction(ActionEvent event) {
398.         // confirmDelete is used to confirm with the user that they want to delete their
    selected RMAs.
399.         boolean confirmDelete = false;
400.
401.         ArrayList<RMAListViewModel> toRemove = new ArrayList<>();
```

```
402.          try {
403.              for (RMAListViewModel model : tblList)
404.                  if (model.getShouldDelete()) {
405.                      if (!confirmDelete) {
406.                          // Ask the user if they are sure they want to delete the RMAs.
407.                          ButtonType confirm = new ButtonType("Confirm",
    ButtonBar.ButtonData.OK_DONE); // Change the button context from OK to Confirm
408.                          Alert a = new Alert(Alert.AlertType.CONFIRMATION,
409.                              "Are you sure you want to permanently delete the selected
    RMAs?",
410.                              confirm, ButtonType.CANCEL
411.                          );
412.
413.                          a.setTitle("Delete RMA(s) Confirmation");
414.                          a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
415.                          Optional<ButtonType> result = a.showAndWait();
416.
417.                          if (result.isPresent() && result.get() != confirm)
418.                              break;
419.                          else
420.                              confirmDelete = true;
421.                      }
422.                      model.delete();
423.                      toRemove.add(model);
424.                  }
425.          } catch (SQLException e) {
426.              Alert a = new Alert(Alert.AlertType.ERROR);
427.              a.setContentText("Error while connecting to SQL Server database!" +
    System.lineSeparator() + e.getLocalizedMessage());
428.              a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
429.              a.show();
430.          } finally {
431.              if (confirmDelete) {
432.                  tblList.removeAll(toRemove);
433.                  btnRefresh.fire();
434.              }
435.          }
436.      }
437.  }
438.
```

LoginModel.java

```
1.  package RMA;
2.
3.  import javafx.beans.property.SimpleStringProperty;
4.  import javafx.beans.property.StringProperty;
5.  import javafx.collections.FXCollections;
6.  import javafx.collections.ObservableList;
7.  import javafx.scene.control.Alert;
8.  import javafx.scene.layout.Region;
9.
10. import java.io.IOException;
11. import java.net.DatagramPacket;
12. import java.net.DatagramSocket;
13. import java.net.InetAddress;
14.
15. /** LoginModel contains the fields necessary to represent the model for the Login form.
16.  */
17. public class LoginModel {
18.     /** user contains the user's entered username.
19.      *  (Requirement 1.1)
20.      */
21.     private final StringProperty user;
22.     /** pass contains the user's entered password.
23.      *  (Requirement 1.2)
24.      */
25.     private final StringProperty pass;
26.     /** role contains the user's resulting role in the RMA database (admins, analysts, or
    engineers).
27.      *  (Requirement 1.3)
28.      */
29.     private final StringProperty role;
30.     /** selectedInstance contains the user's currently selected instance on the Login form.
31.      *  (Requirement 1.4)
32.      */
33.     private final StringProperty selectedInstance;
34.     /** instances contains the SQL Server instances found on the network.
35.      */
36.     private final ObservableList<String> instances;
37.     /** db contains the value for the RMA database name.
38.      */
39.     private final StringProperty db;
40.
41.     /** The default constructor for LoginModel initializes the fields with empty values,
42.      *  and then searches the network for listening SQL Server instances.
43.      */
44.     public LoginModel() {
45.         user = new SimpleStringProperty();
46.         pass = new SimpleStringProperty();
47.         role = new SimpleStringProperty();
48.         instances = FXCollections.observableArrayList();
49.         selectedInstance = new SimpleStringProperty();
50.         db = new SimpleStringProperty("rma");
51.         getInstances();
52.     }
53.
54.     /** getInstances issues a request to any listening SQL Server Browser Service
55.      *  on default UDP port 1434 for their information of running instances, and any
56.      *  that respond are added to the list of instances in the format
    HOSTNAME\INSTANCE_NAME.
57.      *
58.      *  If no hosts respond, an error message is shown and the program exits.
```

```
59.      *  (Requirement 1.4.3)
60.      */
61.     private void getInstances() {
62.         try {
63.             // Broadcast to SQL Server Browsers on network.
64.             DatagramSocket socket = new DatagramSocket();
65.             InetAddress ip = InetAddress.getByName("255.255.255.255");
66.             DatagramPacket packet = new DatagramPacket(new byte[]{2}, 1, ip, 1434);
67.             socket.send(packet);
68.
69.             // Receive response
70.             byte[] buff = new byte[1024 * 1024];
71.             DatagramPacket recv = new DatagramPacket(buff, buff.length);
72.             socket.setSoTimeout(5 * 1000); // in milliseconds
73.             socket.receive(recv);
74.
75.             // Parse response
76.             String response = new String(recv.getData(), 0, recv.getLength());
77.             String[] tokens = response.split(";");
78.
79.             String host = "", instance = "";
80.             for (int i = 0; i < tokens.length; i++)
81.                 if (tokens[i].contains("ServerName"))
82.                     host = tokens[i + 1];
83.                 else if (tokens[i].contentEquals("InstanceName")) {
84.                     instance = tokens[i + 1];
85.                     instances.add(host + "\\" + instance);
86.                 }
87.
88.             // Close socket
89.             socket.close();
90.         } catch (IOException e) {
91.             Alert a = new Alert(Alert.AlertType.ERROR);
92.             a.setContentText(
93.                 "Error while fetching SQL Server instances! No SQL Server Browser instances
    responding." +
94.                 System.lineSeparator() + "Please make sure that SQL Server Browser service
    is running and that UDP port 1434 is not being blocked." +
95.                 System.lineSeparator() + e.getLocalizedMessage()
96.             );
97.             a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
98.             a.showAndWait();
99.             System.exit(1);
100.          }
101.     }
102.
103.     /** getUser returns the String stored in user.
104.      * @return The {@link String} value stored in the user field.
105.      */
106.     public String getUser() {return user.getValueSafe();}
107.
108.     /** userProperty returns the StringProperty field containing the user's username.
109.      * @return The {@link StringProperty} user field.
110.      */
111.     public StringProperty userProperty() {return user;}
112.
113.     /** getPass returns the String stored in pass.
114.      * @return The {@link String} stored in pass.
115.      */
116.     public String getPass() {return pass.getValueSafe();}
117.
118.     /** passProperty returns the StringProperty field containing the user's password.
119.      * @return The {@link StringProperty} pass field.
120.      */
121.     public StringProperty passProperty() {return pass;}
```

```
122.
123.      /** setRole sets the value of role to the user-provided newRole parameter.
124.       *
125.       * @param newRole The role (admin, analyst, or engineer) to store.
126.       */
127.      public void setRole(String newRole) {role.setValue(newRole);}
128.
129.      /** instancesProperty returns the {@link ObservableList} containing the list of
    instances found on the network.
130.       * @return The {@link ObservableList} instances field.
131.       */
132.      public ObservableList<String> instancesProperty() {return instances;}
133.
134.      /** getSelectedInstance returns the String that the user selected from instances.
135.       * @return The {@link String} stored in selectedInstance.
136.       */
137.      public String getSelectedInstance() {return selectedInstance.getValueSafe();}
138.
139.      /** selectedInstanceProperty returns the StringProperty containing the user's
140.       *  selected instance.
141.       * @return The {@link StringProperty} selectedInstance field.
142.       */
143.      public StringProperty selectedInstanceProperty() {return selectedInstance;}
144.
145.      /** getDB returns the {@link String} stored in db.
146.       * @return The {@link String} stored in db.
147.       */
148.      public String getDB() {return db.getValueSafe();}
149.  }
150.
```

RMAListViewModel.java

```
1.  package RMA;
2.
3.  import javafx.beans.property.*;
4.
5.  import java.sql.SQLException;
6.  import java.time.LocalDateTime;
7.
8.  /** RMAListViewModel contains the fields that make up the model for the
9.   *  RMAListView form.
10.  */
11. public class RMAListViewModel {
12.     /** rmaId contains the identifier to the RMA request.
13.      */
14.     private StringProperty rmaId;
15.     /** status contains the textual description of the RMA's status.
16.      */
17.     private StringProperty status;
18.     /** shipReplaceRepair indicates whether this RMA's replacement or repaired part was
    shipped.
19.      */
20.     private BooleanProperty shipReplaceRepair;
21.     /** customerName contains the customer name associated with the RMA request.
22.      */
23.     private StringProperty customerName;
24.     /** businessName contains the business name associated with the RMA request.
25.      */
26.     private StringProperty businessName;
27.     /** address maintains the business address associated with this RMA request.
28.      */
29.     private ObjectProperty<CustomerAddress> address;
30.     /** product contains the product information associated with this RMA request.
31.      */
32.     private ObjectProperty<PurchaseOrderProduct> product;
33.     /** returnQuantity contains the number of items that are being returned.
34.      */
35.     private IntegerProperty returnQuantity;
36.     /** shouldDelete contains whether this RMA request is currently selected for deletion in
    RMAListView.
37.      */
38.     private BooleanProperty shouldDelete;
39.     /** created contains the date and time the RMA was created.
40.      */
41.     private ObjectProperty<LocalDateTime> created;
42.
43.     /** This constructor for RMAListViewModel takes in an RMA ID in order to fetch the
    necessary related information.
44.      * @param newRMAId A {@link String} containing the RMA ID to reference.
45.      * @throws SQLException If there is an issue connecting to the SQL Server database or
    the SQL query.
46.      */
47.     public RMAListViewModel(String newRMAId) throws SQLException {
48.         // Use DBService to fetch the RMA info.
49.         RMAListViewModel model = DBService.getInstance().getRMA(newRMAId);
50.         // Initialize properties.
51.         rmaId = new SimpleStringProperty(model.getRMAId());
52.         status = new SimpleStringProperty(model.getStatus());
53.         shipReplaceRepair = new SimpleBooleanProperty(model.getShipReplaceRepair());
54.         customerName = new SimpleStringProperty(model.getCustomerName());
55.         businessName = new SimpleStringProperty(model.getBusinessName());
56.         address = new SimpleObjectProperty<CustomerAddress>(model.getAddress());
57.         product = new SimpleObjectProperty<PurchaseOrderProduct>(model.getProduct());
```

```
58.         returnQuantity = new SimpleIntegerProperty(model.getReturnQuantity());
59.         shouldDelete = new SimpleBooleanProperty(model.getShouldDelete());
60.         created = new SimpleObjectProperty<LocalDateTime>(model.getCreated());
61.     }
62.
63.     /** Constructor for RMAListViewModel that manually fills in all the field details.
64.      * @param rmaId The {@link String} RMA ID to assign to this RMAListViewModel.
65.      * @param status The {@link String} RMA status description to use for this
    RMAListViewModel.
66.      * @param shipReplaceRepair The boolean status of shipping a replacement or repair.
67.      * @param customerName The {@link String} customer name to use.
68.      * @param businessName The {@link String} business name to use.
69.      * @param address The CustomerAddress to associate with this RMAListViewModel.
70.      * @param product The PurchaseOrderProduct to associate with this RMAListViewModel.
71.      * @param returnQuantity The amount of product being returned by the customer.
72.      * @param shouldDelete Whether this RMA should be flagged for deletion.
73.      * @param created The {@link LocalDateTime} to assign to the RMA in this
    RMAListViewModel.
74.      */
75.     public RMAListViewModel(
76.         String rmaId, String status, boolean shipReplaceRepair,
77.         String customerName, String businessName, CustomerAddress address,
78.         PurchaseOrderProduct product, int returnQuantity, boolean shouldDelete,
79.         LocalDateTime created
80.     ) {
81.         // Initialize properties.
82.         this.rmaId = new SimpleStringProperty(rmaId);
83.         this.status = new SimpleStringProperty(status);
84.         this.shipReplaceRepair = new SimpleBooleanProperty(shipReplaceRepair);
85.         this.customerName = new SimpleStringProperty(customerName);
86.         this.businessName = new SimpleStringProperty(businessName);
87.         this.address = new SimpleObjectProperty<CustomerAddress>(address);
88.         this.product = new SimpleObjectProperty<PurchaseOrderProduct>(product);
89.         this.returnQuantity = new SimpleIntegerProperty(returnQuantity);
90.         this.shouldDelete = new SimpleBooleanProperty(shouldDelete);
91.         this.created = new SimpleObjectProperty<LocalDateTime>(created);
92.     }
93.
94.     /** getRMAId returns the {@link String} value stored in rmaId.
95.      * @return The {@link String} value stored in rmaId.
96.      */
97.     public String getRMAId() {return rmaId.getValueSafe();}
98.
99.     /** rmaIdProperty returns the {@link StringProperty} field containing the RMA ID.
100.       * @return The {@link StringProperty} rmaId field.
101.       */
102.      public StringProperty rmaIdProperty() {return rmaId;}
103.
104.      /** getStatus returns the {@link String} status description stored in status.
105.       * @return The {@link String} value stored in status.
106.       */
107.      public String getStatus() {return status.getValueSafe();}
108.
109.      /** statusProperty returns the {@link StringProperty} field containing the
110.       *  textual description of the RMA status.
111.       * @return The {@link StringProperty} status field.
112.       */
113.      public StringProperty statusProperty() {return status;}
114.
115.      /** getShipReplaceRepair returns the {@link Boolean} value stored in
    shipReplaceRepair.
116.       * @return The {@link Boolean} value stored in shipReplaceRepair.
117.       */
118.      public Boolean getShipReplaceRepair() {return shipReplaceRepair.get();}
119.
```

```
120.      /** shipReplaceRepairProperty returns the {@link BooleanProperty} field containing
121.       *  whether the RMA's replacement or repair was shipped.
122.       * @return The {@link BooleanProperty} shipReplaceRepair field.
123.       */
124.      public BooleanProperty shipReplaceRepairProperty() {return shipReplaceRepair;}
125.
126.      /** getCustomerName returns the {@link String} contained in customerName.
127.       * @return The {@link String} stored in customerName.
128.       */
129.      public String getCustomerName() {return customerName.getValueSafe();}
130.
131.      /** customerNameProperty returns the {@link StringProperty} containing the customer's
    name
132.       *  referenced in the RMA request.
133.       * @return The {@link StringProperty} customerName field.
134.       */
135.      public StringProperty customerNameProperty() {return customerName;}
136.
137.      /** getBusinessName returns the {@link String} value stored in businessName.
138.       * @return The {@link String} stored in businessName.
139.       */
140.      public String getBusinessName() {return businessName.getValueSafe();}
141.
142.      /** businessNameProperty returns the {@link StringProperty} containing the customer's
143.       *  business name referenced in the RMA request.
144.       * @return The {@link StringProperty} businessName field.
145.       */
146.      public StringProperty businessNameProperty() {return businessName;}
147.
148.      /** getAddress returns the CustomerAddress object stored in address.
149.       * @return The CustomerAddress stored in address.
150.       */
151.      public CustomerAddress getAddress() {return address.getValue();}
152.
153.      /** addressProperty returns the {@link ObjectProperty} containing the
154.       *  customer's business address.
155.       * @return The {@link ObjectProperty} address field.
156.       */
157.      public ObjectProperty<CustomerAddress> addressProperty() {return address;}
158.
159.      /** getProduct returns the PurchaseOrderProduct stored in product.
160.       * @return The PurchaseOrderProduct stored in product.
161.       */
162.      public PurchaseOrderProduct getProduct() {return product.getValue();}
163.
164.      /** productProperty returns the {@link ObjectProperty} containing the
165.       *  RMA's referenced PurchaseOrderProduct.
166.       * @return The {@link ObjectProperty} product field.
167.       */
168.      public ObjectProperty<PurchaseOrderProduct> productProperty() {return product;}
169.
170.      /** getReturnQuantity returns the {@link Integer} stored in returnQuantity.
171.       * @return The {@link Integer} stored in returnQuantity.
172.       */
173.      public Integer getReturnQuantity() {return returnQuantity.getValue();}
174.
175.      /** returnQuantityProperty returns the {@link IntegerProperty} containing the number
176.       *  of items that are being returned.
177.       * @return The {@link IntegerProperty} returnQuantity field.
178.        */
179.      public IntegerProperty returnQuantityProperty() {return returnQuantity;}
180.
181.      /** getShouldDelete returns the {@link Boolean} value stored in shouldDelete.
182.       * @return The {@link Boolean} value stored in shouldDelete.
183.        */
```

```
184.      public Boolean getShouldDelete() {return shouldDelete.getValue();}
185.
186.      /** shouldDeleteProperty returns the {@link BooleanProperty} containing whether
187.       *  the user is planning to delete the RMA request referenced by this model.
188.       * @return The {@link BooleanProperty} shouldDelete field.
189.       */
190.      public BooleanProperty shouldDeleteProperty() {return shouldDelete;}
191.
192.      /** delete removes this RMA request from the database.
193.       * @throws SQLException If there is an issue connecting to the database or processing
    the query.
194.       */
195.      public void delete() throws SQLException {
196.          DBService.getInstance().deleteRMA(rmaId.getValueSafe());
197.      }
198.
199.      /** getCreated returns the {@link LocalDateTime} stored in created.
200.       * @return The {@link LocalDateTime} stored in created.
201.       */
202.      public LocalDateTime getCreated() {return created.getValue();}
203.
204.      /** getCreatedProperty returns the {@link ObjectProperty} containing the
205.       *  creation date and time of this RMA request.
206.       * @return The {@link ObjectProperty} created field.
207.       */
208.      public ObjectProperty<LocalDateTime> createdProperty() {return created;}
209.  }
210.
```

CustomerAddress.java

```
1.  package RMA;
2.
3.  import javafx.beans.binding.ObjectBinding;
4.
5.  import java.sql.SQLException;
6.
7.  /** CustomerAddress contains all the necessary details to fully describe a customer's
    business address.
8.   */
9.  public class CustomerAddress extends ObjectBinding<CustomerAddress> {
10.     /** addressId contains the {@link Integer} id for this CustomerAddress.
11.      */
12.     private int addressId;
13.     /** customerName contains the customer name {@link String} associated with this
    CustomerAddress.
14.      */
15.     private String customerName;
16.     /** businessName contains the business name {@link String} associated with this
    CustomerAddress.
17.      */
18.     private String businessName;
19.     /** address1 contains the first part of the street address as a {@link String}.
20.      */
21.     private String address1;
22.     /** address2 is optional and contains the second part of the street address as a {@link
    String}.
23.      */
24.     private String address2;
25.     /** city contains the city {@link String} associated with this CustomerAddress.
26.      */
27.      private String city;
28.     /** county contains the optional county {@link String} associated with this
    CustomerAddress.
29.      */
30.     private String county;
31.     /** stateOrProvince contains the {@link String} state or province initials associated
    with this CustomerName.
32.      */
33.     private String stateOrProvince;
34.     /** zip contains the zip or postal code as a {@link String} for this CustomerAddress.
35.      */
36.     private String zip;
37.     /** country contains the name of the country as a {@link String} for this
    CustomerAddress.
38.      */
39.     private String country;
40.     /** phone contains the phone number for this business, stored as a {@link String} for
    flexibility.
41.      */
42.     private String phone;
43.     /** fax contains the fax number for this business, stored as a {@link String} for
    flexibility.
44.      */
45.     private String fax;
46.     /** showBusinessName is used to only output the business name in the computeValue
    function.
47.      */
48.     private boolean showBusinessName;
```

```
49.     /** EMPTY_ADDRESS is used to output an empty CustomerAddress when needed.
50.      */
51.     public static final CustomerAddress EMPTY_ADDRESS = new
    CustomerAddress(0,"","","","","","","","","","","");
52.
53.     /** Constructor for CustomerAddress that takes the passed-in addressId and fetches the
    address
54.      *  information from the database.
55.      * @param addressId The address ID to search for in the database.
56.      * @throws SQLException If there is an issue connecting to the SQL Server database or
    the SQL query.
57.      */
58.     public CustomerAddress(int addressId) throws SQLException {
59.         // Use DBService to fetch the CustomerAddress info.
60.         CustomerAddress address = DBService.getInstance().getCustomerAddress(addressId);
61.         // Initialize properties.
62.         this.addressId = address.getAddressId();
63.         customerName = address.getCustomerName();
64.         businessName = address.getBusinessName();
65.         address1 = address.getAddress1();
66.         address2 = address.getAddress2();
67.         city = address.getCity();
68.         county = address.getCounty();
69.         stateOrProvince = address.getStateOrProvince();
70.         zip = address.getZip();
71.         country = address.getCountry();
72.         phone = address.getPhone();
73.         fax = address.getFax();
74.         showBusinessName = false;
75.     }
76.
77.     /** Constructor for CustomerAddress that manually fills in all the details.
78.      * @param addressId The int address identifier to assign to this CustomerAddress.
79.      * @param customerName The {@link String} customer name associated with this
    CustomerAddress.
80.      * @param businessName The {@link String} name of the business at this address.
81.      * @param address1 The {@link String} first part of the street address.
82.      * @param address2 The optional {@link String} second part of the street address.
83.      * @param city The {@link String} name of the city.
84.      * @param county The {@link String} name of the county.
85.      * @param stateOrProvince The {@link String} state or province initials.
86.      * @param zip The {@link String} zip or postal code for this address.
87.      * @param country The optional {@link String} name of the country.
88.      * @param phone The {@link String} main phone number for this address.
89.      * @param fax The optional {@link String} fax number for this address.
90.      */
91.     public CustomerAddress(
92.         int addressId, String customerName, String businessName, String address1, String
    address2,
93.         String city, String county, String stateOrProvince, String zip, String country,
    String phone,
94.         String fax
95.     ) {
96.         this.addressId = addressId;
97.         this.customerName = customerName;
98.         this.businessName = businessName;
99.         this.address1 = address1;
100.          this.address2 = address2;
101.          this.city = city;
102.          this.county = county;
103.          this.stateOrProvince = stateOrProvince;
104.          this.zip = zip;
105.          this.country = country;
106.          this.phone = phone;
107.          this.fax = fax;
```

```
108.          showBusinessName = false;
109.      }
110.
111.      /** computeValue returns the value representing this CustomerAddress when bound as an
    {@link javafx.beans.property.ObjectProperty}.
112.       * @return A {@link String} containing either just the business name (if
    showBusinessName is true), or the customer
113.       * name, business name, street, city, county, state, zip, country, phone, and fax (if
    showBusinessName is false).
114.       */
115.      @Override
116.      protected CustomerAddress computeValue() {
117.          CustomerAddress result = new CustomerAddress(addressId, customerName,
    businessName, address1, address2, city,
118.              county, stateOrProvince, zip, country, phone, fax);
119.          result.setShowBusinessName(showBusinessName);
120.          return result;
121.      }
122.
123.      /** toString is used to output the content of this CustomerAddress in the appropriate
    form depending on the value of
124.       *  showBusinessName.
125.       * @return A {@link String} either containing the businessName if showBusinessName is
    true, otherwise the entire
126.       * address if it is false.
127.       */
128.      @Override
129.      public String toString() {
130.          return showBusinessName ? businessName :
131.              (customerName.isEmpty() ? "" : customerName + System.lineSeparator()) +
132.              (businessName.isEmpty() ? "" : businessName + System.lineSeparator()) +
133.              (address1.isEmpty() ? "" : address1 + System.lineSeparator()) +
134.              (address2.isEmpty() ? "" : address2 + System.lineSeparator()) +
135.              (city.isEmpty() ? "" : city + System.lineSeparator()) +
136.              (county.isEmpty() ? "" : county + System.lineSeparator()) +
137.              (stateOrProvince.isEmpty() ? "" : stateOrProvince + System.lineSeparator()) +
138.              (zip.isEmpty() ? "" : zip + System.lineSeparator()) +
139.              (country.isEmpty() ? "" : country + System.lineSeparator()) +
140.              (phone.isEmpty() ? "" : "Phone: " + phone + System.lineSeparator()) +
141.              (fax.isEmpty() ? "" : "Fax: " + fax + System.lineSeparator())
142.          ;
143.      }
144.
145.      /** getAddressId returns the address ID stored in this CustomerAddress.
146.       * @return The int addressId field.
147.       */
148.      public int getAddressId() {return addressId;}
149.
150.      /** getCustomerName returns the customer name stored in this CustomerAddress.
151.       * @return The {@link String} customerName field.
152.       */
153.      public String getCustomerName() {return customerName;}
154.
155.      /** getBusinessName returns the business name stored in this CustomerAddress.
156.       * @return The {@link String} businessName field.
157.       */
158.      public String getBusinessName() {return businessName;}
159.
160.      /** getAddress1 returns the first part of the street address stored in this
    CustomerAddress.
161.       * @return The {@link String} address1 field.
162.       */
163.      public String getAddress1() {return address1;}
164.
```

```
165.      /** getAddress2 returns the second, optional part of the street address stored in this
    CustomerAddress.
166.       * @return The {@link String} address2 field.
167.       */
168.      public String getAddress2() {return address2;}
169.
170.      /** getCity returns the name of the city stored in this CustomerAddress.
171.       * @return The {@link String} city field.
172.       */
173.      public String getCity() {return city;}
174.
175.      /** getCounty returns the name of the county stored in this CustomerAddress.
176.       * @return The {@link String} county field.
177.       */
178.      public String getCounty() {return county;}
179.
180.      /** getStateOrProvince returns the initials for the state or province stored in this
    CustomerAddress.
181.       * @return The {@link String} stateOrProvince field.
182.       */
183.      public String getStateOrProvince() {return stateOrProvince;}
184.
185.      /** getZip returns the zip or postal code stored in this CustomerName.
186.       * @return The {@link String} zip field.
187.       */
188.      public String getZip() {return zip;}
189.
190.      /** getCountry returns the country name stored in this CustomerAddress.
191.       * @return The {@link String} country field.
192.       */
193.      public String getCountry() {return country;}
194.
195.      /** getPhone returns the phone number stored in this CustomerAddress.
196.       * @return The {@link String} phone field.
197.       */
198.      public String getPhone() {return phone;}
199.
200.      /** getFax returns the fax number stored in this CustomerAddress.
201.       * @return The {@link String} fax field.
202.        */
203.      public String getFax() {return fax;}
204.
205.      /** setShowBusinessName is only used if the programmer wants only the business name to
    show in the computeValue function.
206.       * @param newShowBusinessName The new boolean to set for showBusinessName (whether we
    want only the name showing or
207.       *                            the entire address).
208.       */
209.      public void setShowBusinessName(boolean newShowBusinessName) {showBusinessName =
    newShowBusinessName;}
210.
211.      /** createShippingAddress is used to create a copy of this CustomerAddress, set to
    show the entire address, for the
212.       *  shippingAddress field.
213.       * @return A new instance of this {@link CustomerAddress} with showBusinessName set to
    false.
214.       */
215.      public CustomerAddress createShippingAddress() {
216.          return new CustomerAddress(addressId, customerName, businessName, address1,
    address2, city, county,
217.              stateOrProvince, zip, country, phone, fax);
218.      }
219.  }
220.
```

PurchaseOrderProduct.java

```
1.  package RMA;
2.
3.  import javafx.beans.binding.ObjectBinding;
4.
5.  import java.sql.SQLException;
6.  import java.time.LocalDate;
7.
8.  /** PurchaseOrderProduct contains the specific information about a product
9.   *  coming from a purchase order in the RMA database.
10.  */
11. public class PurchaseOrderProduct extends ObjectBinding<PurchaseOrderProduct> {
12.     /** purchaseOrderProductId contains the unique integer value identifying this
    PurchaseOrderProduct.
13.      */
14.     private int purchaseOrderProductId;
15.     /** poNumber contains the business address's PO number {@link String} tied to this
    PurchaseOrderProduct.
16.      */
17.     private String poNumber;
18.     /** productName contains the product's name as a {@link String}.
19.      */
20.     private String productName;
21.     /** productCategory contains the product's category name as a {@link String}.
22.      */
23.     private String categoryName;
24.     /** quantity contains the integer value of how many units of the product the customer is
    returning.
25.      */
26.     private int quantity;
27.     /** orderDate contains the date the customer ordered this product, stored as a {@link
    LocalDate}.
28.      */
29.     private LocalDate orderDate;
30.     /** deliverDate contains the date the customer received this product, stored as a {@link
    LocalDate}.
31.      */
32.     private LocalDate deliverDate;
33.
34.     /** Constructor that takes the passed-in purchaseOrderProductId and fetches the
    product's
35.      *  information from the database.
36.      * @param purchaseOrderProductId The id to search for in the database.
37.      * @throws SQLException If there is an issue connecting to the SQL Server database or
    the SQL query.
38.      */
39.     public PurchaseOrderProduct(int purchaseOrderProductId) throws SQLException {
40.         // Fetch product from database.
41.         PurchaseOrderProduct product =
    DBService.getInstance().getPurchaseOrderProduct(purchaseOrderProductId);
42.         // Populate the fields.
43.         this.purchaseOrderProductId = product.getPurchaseOrderProductId();
44.         poNumber = product.getPONumber();
45.         productName = product.getProductName();
46.         categoryName = product.getProductCategory();
47.         quantity = product.getQuantity();
48.         orderDate = product.getOrderDate();
49.         deliverDate = product.getDeliverDate();
50.     }
51.
52.     /** Constructor for PurchaseOrderProduct that manually fills in all the details.
```

```
53.      * @param purchaseOrderProductId The int unique id to store in this
    PurchaseOrderProduct.
54.      * @param poNumber The {@link String} PO number associated with this
    PurchaseOrderProduct.
55.      * @param productName The {@link String} product's name.
56.      * @param categoryName The {@link String} category name in which this product resides.
57.      * @param quantity The int amount being ordered.
58.      * @param orderDate The {@link LocalDate} of when the product was ordered by the
    customer.
59.      * @param deliverDate The {@link LocalDate} of when the product was delivered to the
    customer.
60.      */
61.     public PurchaseOrderProduct(
62.         int purchaseOrderProductId, String poNumber, String productName,
63.         String categoryName, int quantity, LocalDate orderDate,
64.         LocalDate deliverDate
65.     ) {
66.         // Populate the fields with the passed-in parameters.
67.         this.purchaseOrderProductId = purchaseOrderProductId;
68.         this.poNumber = poNumber;
69.         this.productName = productName;
70.         this.categoryName = categoryName;
71.         this.quantity = quantity;
72.         this.orderDate = orderDate;
73.         this.deliverDate = deliverDate;
74.     }
75.
76.     /** computeValue returns the value representing this PurchaseOrderProduct when bound as
    an {@link javafx.beans.property.ObjectProperty}.
77.      * @return A copy of this PurchaseOrderProduct {@link String} containing the category
    name, then product name.
78.      */
79.     @Override
80.     protected PurchaseOrderProduct computeValue() {
81.         return new PurchaseOrderProduct(
82.             purchaseOrderProductId,
83.             poNumber,
84.             productName,
85.             categoryName,
86.             quantity,
87.             orderDate,
88.             deliverDate
89.         );
90.     }
91.
92.     /** toString is used to output the appropriate value to represent this
    PurchaseOrderProduct.
93.      * @return A {@link String} containing the category name, a comma, and the product name.
94.      */
95.     @Override
96.     public String toString() {return String.format("%s, %s", categoryName, productName);}
97.
98.     /** getPurchaseOrderProductId returns the id of this PurchaseOrderProduct.
99.      * @return An int, contained in the purchaseOrderProductId field.
100.       */
101.      public int getPurchaseOrderProductId() {return purchaseOrderProductId;}
102.
103.      /** getPONumber returns the purchase order number associated with this
    PurchaseOrderProduct.
104.       * @return The {@link String} poNumber field.
105.       */
106.      public String getPONumber() {return poNumber;}
107.
108.      /** getProductName returns the product name contained in the productName {@link
    String} field.
```

```
109.        * @return The {@link String} value of the productName field.
110.        */
111.       public String getProductName() {return productName;}
112.
113.       /** getProductCategory returns the category name contained in the productCategory
    {@link String} field.
114.        * @return The {@link String} value of the productCategory field.
115.        */
116.       public String getProductCategory() {return categoryName;}
117.
118.       /** getQuantity returns the int quantity of the amount being returned by the customer.
119.        * @return The int quantity field.
120.        */
121.       public int getQuantity() {return quantity;}
122.
123.       /** getOrderDate returns the date the customer placed the order for the item, stored
    in a {@link LocalDate}.
124.        * @return The {@link LocalDate} orderDate field.
125.        */
126.       public LocalDate getOrderDate() {return orderDate;}
127.
128.       /** getDeliverDate returns the date the product was delivered to the customer, stored
    in a {@link LocalDate}.
129.        * @return The {@link LocalDate} deliverDate field.
130.        */
131.       public LocalDate getDeliverDate() {return deliverDate;}
132.   }
133.
```

RMAFomController.java

```
1.  package RMA;
2.
3.  import javafx.application.Platform;
4.  import javafx.beans.value.ChangeListener;
5.  import javafx.beans.value.ObservableValue;
6.  import javafx.event.ActionEvent;
7.  import javafx.event.Event;
8.  import javafx.fxml.FXML;
9.  import javafx.fxml.Initializable;
10. import javafx.scene.control.*;
11. import javafx.scene.layout.Region;
12. import javafx.util.converter.NumberStringConverter;
13.
14. import java.net.URL;
15. import java.sql.SQLException;
16. import java.util.ArrayList;
17. import java.util.Optional;
18. import java.util.ResourceBundle;
19.
20. /** RMAFormController contains the logic and field references to the GUI controls on the
    RMAForm.
21.  */
22. public class RMAFormController implements Initializable {
23.     /**
24.      * lblRMANumberValue displays the assigned RMA number on save.
25.      */
26.     @FXML
27.     private Label lblRMANumberValue;
28.     /**
29.      * cmbCustomerName contains list of customer names in the database.
30.      * (Requirement 4.1)
31.      */
32.     @FXML
33.     private ComboBox<String> cmbCustomerName;
34.     /**
35.      * cmbBusinessName contains the list of business names for the selected customer name in
    the database.
36.      * (Requirement 4.2)
37.      */
38.     @FXML
39.     private ComboBox<CustomerAddress> cmbBusinessName;
40.     /**
41.      * cmbPoNumber contains the PO numbers associated with the selected business name.
42.      * (Requirement 4.11)
43.      */
44.     @FXML
45.     private ComboBox<String> cmbPoNumber;
46.     /**
47.      * txtCustomerInfoShip is used to show the shipping address associated with the selected
    business,
48.      * and auto populates when business name is selected.
49.      * (Requirement 4.12)
50.      */
51.     @FXML
52.     private TextArea txtShippingAddress;
53.     /**
54.      * lblOwnerValue shows the employee username assigned to the RMA upon saving.
55.      */
56.     @FXML
57.     private Label lblOwnerValue;
58.     /**
```

```
59.      * cmbReasonCode contains the list of return reason codes in the database.
60.      * (Requirement 4.3)
61.      */
62.     @FXML
63.     private ComboBox<String> cmbReasonCode;
64.     /**
65.      * cmbCreditReplaceRepair contains the values Credit, Replace, and Repair, used to
    determine which action we
66.      * will be taking on the return.
67.      * (Requirement 4.4)
68.      */
69.     @FXML
70.     private ComboBox<String> cmbCreditReplaceRepair;
71.     /**
72.      * cmbRMAStatus contains list of RMA Statuses in the database.
73.      * (Requirement 4.5)
74.      */
75.     @FXML
76.     private ComboBox<String> cmbRMAStatus;
77.     /**
78.      * txtAddInfoSpecialInt is used to store any additional instructions about the return
    entered by the user.
79.      * (Requirement 4.13)
80.      */
81.     @FXML
82.     private TextArea txtAddInfoSpecialInt;
83.     /**
84.      * cmbProduct contains the list of {@link PurchaseOrderProduct}s for the chosen PO
    number in the database.
85.      * (Requirement 4.7)
86.      */
87.     @FXML
88.     private ComboBox<PurchaseOrderProduct> cmbProduct;
89.     /**
90.      * txtReturnLabelTrack is used to store the return label tracking number.
91.      * (Requirement 4.7.2)
92.      */
93.     @FXML
94.     private TextField txtReturnLabelTrack;
95.     /**
96.      * txtQuantity is used to enter the customer's return quantity.
97.      * (Requirement 4.7.3)
98.      */
99.     @FXML
100.      private TextField txtQuantity;
101.      /**
102.       * txtInitialEvaluation allows the user to enter in their initial evaluation of the
    returned product(s).
103.       * (Requirement 4.8)
104.       */
105.      @FXML
106.      private TextArea txtInitialEvaluation;
107.      /**
108.       * cmbDisposition contains the list of disposition in the database.
109.       * (Requirement 4.9)
110.       */
111.      @FXML
112.      private ComboBox<String> cmbDisposition;
113.      /**
114.       * txtDispositionNotes allows the user to enter in notes about the chosen disposition.
115.       * (Requirement 4.9.2)
116.       */
117.      @FXML
118.      private TextArea txtDispositionNotes;
119.      /**
```

```
120.       * txtReplaceRepairTracking allow to enter the tracking number for the replacement or
    repair being shipped
121.       * back to the customer.
122.       * (Requirement 4.14)
123.       */
124.      @FXML
125.      private TextField txtReplaceRepairTracking;
126.      /**
127.       * dtpkrReplaceRepairShipDate allows the user to choose the date when we will be
    shipping
128.       * back the replacement or repair.
129.       * Requirement 4.15)
130.       */
131.      @FXML
132.      private DatePicker dtpkrReplaceRepairShipDate;
133.      /**
134.       * cbShipIndicator indicates that the return/replacement has been shipped.
135.       * (Requirement 4.16)
136.       */
137.      @FXML
138.      private CheckBox cbShipIndicator;
139.      /**
140.       * btnSave saves the RMA to the database.
141.       * (Requirement 4.10)
142.       */
143.      @FXML
144.      private Button btnSave;
145.      /**
146.       * btnSaveClose saves the RMA to the database and then closes the form.
147.       * (Requirement 4.10)
148.       */
149.      @FXML
150.      private Button btnSaveClose;
151.      /**
152.       * btnCancel closes the form without saving.
153.       * (Requirement 4.10)
154.       */
155.      @FXML
156.      private Button btnCancel;
157.      /**
158.       * model is the data model for the RMAForm.
159.       */
160.      private RMAFormModel model;
161.      /**
162.       * customerConnectionError is used to track whether the user had a connection error
    while selecting a value for the Customer
163.       * so that selecting the same value will allow it to try again.
164.       * (Requirement 4.1.2)
165.       */
166.      private boolean customerConnectionError;
167.      /** customerRevert is used to revert back to the old Customer selection without
    prompting again.
168.       */
169.      private boolean customerRevert;
170.      /**
171.       * businessConnectionError is used to track whether the user had a connection error
    while selecting a value for the Business,
172.       * so that selecting the same value will allow it to try again.
173.       * (Requirement 4.2.2)
174.       */
175.      private boolean businessConnectionError;
176.      /** businessRevert is used to revert back to the old Business selection without
    prompting again.
177.       */
178.      private boolean businessRevert;
```

```
179.      /**
180.       * poConnectionError is used to track whether the user had a connection error while
    selecting a value for the PO Number,
181.       * so that selecting the same value will allow it to try again.
182.       * (Requirement 4.11.3)
183.       */
184.      private boolean poConnectionError;
185.      /** poRevert is used to revert back to the old PO Number selection without prompting
    again.
186.       */
187.      private boolean poRevert;
188.
189.
190.      /** Default empty constructor, used when {@link javafx.fxml.FXMLLoader} loads the fxml
    file.
191.       */
192.      public RMAFormController() {}
193.
194.      /**
195.       * initialize ties together the GUI's elements with the RMAFormModel and
196.       * creates any necessary {@link ChangeListener}s on the controls.
197.       * (Requirement 4.1.1, 4.2.1, 4.4.1, 4.4.1.1, 4.7.1, 4.7.3.1, 4.9.1, 4.11.1, 4.11.2,
    4.12.1,
198.       *  4.18, 4.18.1, 4.18.2., 4.18.3, 4.18.4)
199.       *
200.       * @param url The {@link URL} location of the fxml file that describes
201.       *            the RMAForm window elements and their layout.
202.       * @param rb  {@link ResourceBundle} that contains locale-specific data.
203.       */
204.      @Override
205.      public void initialize(URL url, ResourceBundle rb) {
206.          // First initialize a new RMAFormModel.
207.          model = new RMAFormModel();
208.
209.          // Create TextFormatter for txtQuantity to restrict input.
210.          TextFormatter<String> textFormatter = new TextFormatter<>(change -> {
211.              if (!change.isContentChange())
212.                  return change;
213.
214.              String text = change.getControlNewText();
215.
216.              if (!text.matches("^\\d*"))
217.                  return null;
218.              else
219.                  return change;
220.          });
221.
222.          // Bind the model to the UI elements.
223.          lblRMANumberValue.textProperty().bindBidirectional(model.rmaIDProperty());
224.          cmbCustomerName.setItems(model.customerNamesProperty());
225.
    cmbCustomerName.valueProperty().bindBidirectional(model.selectedCustomerNameProperty());
226.          cmbBusinessName.setItems(model.businessNamesProperty());
227.
    cmbBusinessName.valueProperty().bindBidirectional(model.selectedBusinessNameProperty());
228.          cmbBusinessName.setDisable(true);
229.          cmbPoNumber.setItems(model.poNumbersProperty());
230.          cmbPoNumber.valueProperty().bindBidirectional(model.selectedPONumberProperty());
231.          cmbPoNumber.setDisable(true);
232.
    txtShippingAddress.textProperty().bind(model.shippingAddressProperty().asString());
233.          txtShippingAddress.setEditable(false);
234.          cmbReasonCode.setItems(model.returnReasonCodesProperty());
235.
    cmbReasonCode.valueProperty().bindBidirectional(model.selectedReturnReasonCodeProperty());
```

```
236.          cmbCreditReplaceRepair.setItems(model.creditReplaceRepairProperty());
237.
    cmbCreditReplaceRepair.valueProperty().bindBidirectional(model.selectedCreditReplaceRepairPr
    operty());
238.          cmbRMAStatus.setItems(model.rmaStatusesProperty());
239.          cmbRMAStatus.valueProperty().bindBidirectional(model.selectedRMAStatusProperty());
240.
    txtAddInfoSpecialInt.textProperty().bindBidirectional(model.additionalInfoProperty());
241.          cmbProduct.setItems(model.productsProperty());
242.          cmbProduct.valueProperty().bindBidirectional(model.selectedProductProperty());
243.          cmbProduct.setDisable(true);
244.
    txtReturnLabelTrack.textProperty().bindBidirectional(model.returnLabelTrackerProperty());
245.          txtQuantity.textProperty().bindBidirectional(model.returnQuantityProperty(), new
    NumberStringConverter());
246.          txtQuantity.setTextFormatter(textFormatter);
247.
    txtInitialEvaluation.textProperty().bindBidirectional(model.initialEvaluationProperty());
248.          cmbDisposition.setItems(model.dispositionsProperty());
249.
    cmbDisposition.valueProperty().bindBidirectional(model.selectedDispositionProperty());
250.
    txtDispositionNotes.textProperty().bindBidirectional(model.dispositionNotesProperty());
251.
    txtReplaceRepairTracking.textProperty().bindBidirectional(model.replacementTrackingNumberPro
    perty());
252.
    dtpkrReplaceRepairShipDate.valueProperty().bindBidirectional(model.replacementShipDateProper
    ty());
253.
    cbShipIndicator.selectedProperty().bindBidirectional(model.shipReplacementRepairProperty());
254.          customerConnectionError = false;
255.          customerRevert = false;
256.          businessConnectionError = false;
257.          businessRevert = false;
258.          poConnectionError = false;
259.          poRevert = false;
260.
261.
262.          // Add ChangeListeners to Customer, Business Name, PO Number, and
    CreditReplaceRepair ComboBoxes.
263.          cmbCustomerName.getSelectionModel().selectedItemProperty().addListener(
264.                  new ChangeListener<String>() {
265.                      /** changed checks the new and previous value to decide whether to
    clear the Business Name,
266.                       *  PO Number, Shipping Address info, and Product after picking a
    Customer Name.
267.                       * @param observableValue The {@link ObservableValue} variable that
    changed.
268.                       * @param oldValue The old {@link String} value of the {@link
    ObservableValue}.
269.                       * @param newValue The new {@link String} value of the {@link
    ObservableValue}.
270.                       */
271.                      @Override
272.                      public void changed(ObservableValue<? extends String> observableValue,
    String oldValue, String newValue) {
273.                          if (newValue != null && oldValue != null) { // They chose another
    Customer Name.
274.                              if (!newValue.equals(oldValue)) { // They chose a different
    Customer Name.
275.                                  // Confirm if user wants to reset their choices.
276.                                  if (!customerRevert && model.getSelectedBusinessName() !=
    null) { // They selected a Business Name before selecting a different Customer Name.
```

```
277.                                     ButtonType confirm = new ButtonType("Confirm",
    ButtonBar.ButtonData.OK_DONE); // Change the button text from OK to Confirm
278.                                     Alert a = new Alert(
279.                                             Alert.AlertType.CONFIRMATION,
280.                                             "Business Name, PO Number, Shipping Address,
    and Product will be cleared! " +
281.                                                     "Would you like to continue?",
282.                                             confirm,
283.                                             ButtonType.CANCEL
284.                                     );
285.                                     a.setTitle("Reset Business Name, PO Number, Shipping
    Address, and Product");
286.                                     a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
287.                                     Optional<ButtonType> result = a.showAndWait();
288.
289.                                     if (result.isPresent() && result.get() == confirm) {
290.                                         // Clear it backwards so that other change
    listeners don't fire.
291.                                         model.setSelectedProduct(null);
292.                                         model.clearProducts();
293.                                         cmbProduct.setDisable(true);
294.
295.
    model.setShippingAddress(CustomerAddress.EMPTY_ADDRESS);
296.
297.                                         model.setSelectedPONumber(null);
298.                                         model.clearPONumbers();
299.                                         cmbPoNumber.setDisable(true);
300.                                         poConnectionError = false;
301.
302.                                         model.setSelectedBusinessName(null);
303.                                         model.clearBusinessNames();
304.                                         try { // Populate cmbBusinessName
305.
    model.businessNamesProperty().setAll(DBService.getInstance().getCustomerBusinessNames(newVal
    ue));
306.                                             customerConnectionError = false;
307.                                         } catch (SQLException e) {
308.                                             Alert error = new
    Alert(Alert.AlertType.ERROR);
309.                                             error.setContentText(
310.                                                     "Error while connecting to SQL Server
    database!" + System.lineSeparator() +
311.                                                             e.getLocalizedMessage() +
    System.lineSeparator() +
312.                                                             "Please select the value again
    to try again."
313.                                             );
314.
    error.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
315.                                             error.showAndWait();
316.                                             customerConnectionError = true;
317.                                         } finally {
318.                                             businessConnectionError = false;
319.                                         }
320.                                     } else { // Reset the value to oldValue
321.                                         customerRevert = true;
322.                                         model.setSelectedCustomerName(oldValue);
323.                                     }
324.                                 } else if (!customerRevert) { // They selected another
    Customer Name and did not select a Business Name beforehand.
325.                                     model.clearBusinessNames();
326.                                     try { // Populate cmbBusinessName
```

```
327.
    model.businessNamesProperty().setAll(DBService.getInstance().getCustomerBusinessNames(newVal
    ue));
328.                                         customerConnectionError = false;
329.                                     } catch (SQLException e) {
330.                                         Alert error = new Alert(Alert.AlertType.ERROR);
331.                                         error.setContentText(
332.                                             "Error while connecting to SQL Server
    database!" + System.lineSeparator() +
333.                                             e.getLocalizedMessage() +
    System.lineSeparator() +
334.                                             "Please select the value again to try again."
335.                                         );
336.
    error.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
337.                                         error.showAndWait();
338.                                         customerConnectionError = true;
339.                                     }
340.                                 } else // Let change go through and reset customerRevert.
341.                                     customerRevert = false;
342.                             }
343.                         } else if (newValue != null) { // They chose a Customer Name and
    oldValue was null.
344.                             try { // Populate cmbBusinessName
345.
    model.businessNamesProperty().setAll(DBService.getInstance().getCustomerBusinessNames(newVal
    ue));
346.                                 cmbBusinessName.setDisable(false);
347.                                 customerConnectionError = false;
348.                             } catch (SQLException e) {
349.                                 Alert a = new Alert(Alert.AlertType.ERROR);
350.                                 a.setContentText(
351.                                         "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
352.                                                 e.getLocalizedMessage() +
    System.lineSeparator() +
353.                                                 "Please select the value again to try
    again."
354.                                 );
355.                                 a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
356.                                 a.showAndWait();
357.                                 customerConnectionError = true;
358.                             }
359.                         }
360.                     }
361.                 }
362.         );
363.
364.         cmbBusinessName.getSelectionModel().selectedItemProperty().addListener(
365.                 new ChangeListener<CustomerAddress>() {
366.                     /** changed checks the new and previous value to decide whether to
    clear PO Number,
367.                      *  Shipping Address info, and Product after picking a Business Name.
368.                      * @param observableValue The {@link ObservableValue} variable that
    changed.
369.                      * @param oldValue The old {@link String} value of the {@link
    ObservableValue}.
370.                      * @param newValue The new {@link String} value of the {@link
    ObservableValue}.
371.                      */
372.                     @Override
373.                     public void changed(ObservableValue<? extends CustomerAddress>
    observableValue, CustomerAddress oldValue, CustomerAddress newValue) {
374.                         if (newValue == null && oldValue != null) { // The value is being
    programmatically set, continue.
```

```
375.                                 if (model.getSelectedCustomerName() == null) { // The entire
    form is being cleared.
376.                                     model.clearBusinessNames();
377.                                     cmbBusinessName.setDisable(true);
378.                                     model.setShippingAddress(CustomerAddress.EMPTY_ADDRESS);
379.                                 } else { // The customer name is being changed.
380.                                     try { // Populate cmbBusinessName
381.                                         model.clearBusinessNames();
382.                                         model.setSelectedBusinessName(null);
383.
    model.businessNamesProperty().setAll(DBService.getInstance().getCustomerBusinessNames(model.
    getSelectedCustomerName()));
384.                                         cmbBusinessName.setDisable(false);
385.                                         businessConnectionError = false;
386.                                         customerConnectionError = false;
387.                                     } catch (SQLException e) {
388.                                         Alert a = new Alert(Alert.AlertType.ERROR);
389.                                         a.setContentText(
390.                                             "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
391.                                             e.getLocalizedMessage() + System.lineSeparator() +
392.                                             "Please select the value again to try again."
393.                                         );
394.                                         a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
395.                                         a.showAndWait();
396.                                         businessConnectionError = true;
397.                                         customerConnectionError = true;
398.                                     }
399.                                 }
400.                             } else if (newValue != null && oldValue != null) { // They chose
    another Business Name.
401.                                 if (!newValue.equals(oldValue)) { // They chose a different
    Business Name.
402.                                     // Confirm if user wants to reset their choices.
403.                                     if (!businessRevert &&
    !model.getSelectedPONumber().isEmpty()) {  // The user selected a PO Number before selecting
    a different Business Name.
404.                                         ButtonType confirm = new ButtonType("Confirm",
    ButtonBar.ButtonData.OK_DONE); // Change the button text from OK to Confirm
405.                                         Alert a = new Alert(
406.                                             Alert.AlertType.CONFIRMATION,
407.                                             "PO Number and Product will be cleared! Would you
    like to continue?",
408.                                             confirm,
409.                                             ButtonType.CANCEL
410.                                         );
411.                                         a.setTitle("Reset PO Number and Product");
412.                                         a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
413.                                         Optional<ButtonType> result = a.showAndWait();
414.
415.                                         if (result.isPresent() &&
    result.get().equals(confirm)) {
416.                                             // Clear it backwards so that other change
    listeners don't fire.
417.                                             model.setSelectedProduct(null);
418.                                             model.clearProducts();
419.                                             cmbProduct.setDisable(true);
420.
421.
    model.setShippingAddress(newValue.createShippingAddress());
422.
423.                                         model.setSelectedPONumber(null);
424.                                         model.clearPONumbers();
425.                                         try { // Populate cmbPoNumber
```

```
426.
    model.poNumbersProperty().setAll(DBService.getInstance().getCustomerAddressPONumbers(model.g
    etSelectedBusinessName()));
427.                                             businessConnectionError = false;
428.                                         } catch (SQLException e) {
429.                                             Alert error = new
    Alert(Alert.AlertType.ERROR);
430.                                             error.setContentText(
431.                                                     "Error while connecting to SQL Server
    database!" + System.lineSeparator() +
432.                                                             e.getLocalizedMessage() +
    System.lineSeparator() +
433.                                                             "Please select the value again
    to try again."
434.                                             );
435.
    error.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
436.                                             error.showAndWait();
437.                                             businessConnectionError = true;
438.                                         } finally {
439.                                             poConnectionError = false;
440.                                         }
441.                                     } else { // Reset to the old value.
442.                                         businessRevert = true;
443.                                         model.setSelectedBusinessName(oldValue);
444.                                     }
445.                                 } else if (!businessRevert) { // User selected another
    Business Name and did not select a PO Number beforehand.
446.
    model.setShippingAddress(newValue.createShippingAddress());
447.                                     model.clearPONumbers();
448.                                     try { // Populate cmbPoNumber
449.
    model.poNumbersProperty().setAll(DBService.getInstance().getCustomerAddressPONumbers(model.g
    etSelectedBusinessName()));
450.                                         businessConnectionError = false;
451.                                     } catch (SQLException e) {
452.                                         Alert a = new Alert(Alert.AlertType.ERROR);
453.                                         a.setContentText(
454.                                                 "Error while connecting to SQL Server
    database!" + System.lineSeparator() +
455.                                                         e.getLocalizedMessage() +
    System.lineSeparator() +
456.                                                         "Please select the value again to
    try again."
457.                                         );
458.
    a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
459.                                         a.showAndWait();
460.                                         businessConnectionError = true;
461.                                     }
462.                                 } else // Let change go through and reset businessRevert
463.                                     businessRevert = false;
464.                             }
465.                         } else if (newValue != null) { // They chose a Business Name and
    oldValue was null.
466.                             model.setShippingAddress(newValue.createShippingAddress());
467.                             try { // Populate cmbPoNumber
468.
    model.poNumbersProperty().setAll(DBService.getInstance().getCustomerAddressPONumbers(model.g
    etSelectedBusinessName()));
469.                                 cmbPoNumber.setDisable(false);
470.                                 businessConnectionError = false;
471.                             } catch (SQLException e) {
472.                                 Alert a = new Alert(Alert.AlertType.ERROR);
```

```
473.                                 a.setContentText(
474.                                         "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
475.                                                 e.getLocalizedMessage() +
    System.lineSeparator() +
476.                                                 "Please select the value again to try
    again."
477.                                 );
478.                                 a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
479.                                 a.showAndWait();
480.                                 businessConnectionError = true;
481.                             }
482.                         }
483.                     }
484.                 }
485.         );
486.
487.         cmbPoNumber.getSelectionModel().selectedItemProperty().addListener(
488.                 new ChangeListener<String>() {
489.                     /** changed checks the new and previous value to decide whether to
    clear Product after picking a PO Number.
490.                      * @param observableValue The {@link ObservableValue} variable that
    changed.
491.                      * @param oldValue The old {@link String} value of the {@link
    ObservableValue}.
492.                      * @param newValue The new {@link String} value of the {@link
    ObservableValue}.
493.                      */
494.                     @Override
495.                     public void changed(ObservableValue<? extends String> observableValue,
    String oldValue, String newValue) {
496.                         if (newValue == null && oldValue != null) { // The value is being
    programmatically set, continue.
497.                             if (model.getSelectedBusinessName() == null) { // The form is
    being cleared or business name was changed.
498.                                 model.clearPONumbers();
499.                                 cmbPoNumber.setDisable(true);
500.
501.                                 model.setSelectedProduct(null);
502.                                 model.clearProducts();
503.                                 cmbProduct.setDisable(true);
504.
505.                                 poConnectionError = false;
506.                             } else { // The user selected another business name.
507.                                 try { // Populate cmbPoNumber
508.                                     model.setSelectedProduct(null);
509.                                     model.clearProducts();
510.                                     cmbProduct.setDisable(true);
511.
512.
    model.poNumbersProperty().setAll(DBService.getInstance().getCustomerAddressPONumbers(model.g
    etSelectedBusinessName()));
513.                                     cmbPoNumber.setDisable(false);
514.                                     businessConnectionError = false;
515.                                     poConnectionError = false;
516.                                 } catch (SQLException e) {
517.                                     Alert a = new Alert(Alert.AlertType.ERROR);
518.                                     a.setContentText(
519.                                         "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
520.                                         e.getLocalizedMessage() + System.lineSeparator() +
521.                                         "Please select the value again to try again."
522.                                     );
523.                                     a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
524.                                     a.showAndWait();
```

```
525.                                    businessConnectionError = true;
526.                                    poConnectionError = true;
527.                                }
528.                            }
529.                        } else if (newValue != null && oldValue != null) { // They chose
    another PO nNumber.
530.                            if (!newValue.equals(oldValue)) { // They chose a different PO
    number.
531.                                // Confirm if user wants to reset their choices.
532.                                if (!poRevert && model.getSelectedProduct() != null) { //
    The user selected a Product before selecting a different PO Number
533.                                    ButtonType confirm = new ButtonType("Confirm",
    ButtonBar.ButtonData.OK_DONE); // Change the button text from OK to Confirm
534.                                    Alert a = new Alert(
535.                                            Alert.AlertType.CONFIRMATION,
536.                                            "Product will be cleared! Would you like to
    continue?",
537.                                            confirm,
538.                                            ButtonType.CANCEL
539.                                    );
540.                                    a.setTitle("Reset Product");
541.                                    a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
542.                                    Optional<ButtonType> result = a.showAndWait();
543.
544.                                    if (result.isPresent() && result.get() == confirm) {
545.                                        model.setSelectedProduct(null);
546.                                        model.clearProducts();
547.                                        try { // Populate cmbProducts
548.
    model.productsProperty().setAll(DBService.getInstance().getPurchaseOrderProducts(newValue));
549.                                            poConnectionError = false;
550.                                        } catch (SQLException e) {
551.                                            Alert error = new
    Alert(Alert.AlertType.ERROR);
552.                                            error.setContentText(
553.                                                    "Error while connecting to SQL Server
    database!" + System.lineSeparator() +
554.                                                            e.getLocalizedMessage() +
    System.lineSeparator() +
555.                                                            "Please select the value again
    to try again."
556.                                            );
557.
    error.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
558.                                            error.showAndWait();
559.                                            poConnectionError = true;
560.                                        }
561.                                    } else { // Set PO Number back to oldValue
562.                                        poRevert = true;
563.                                        model.setSelectedPONumber(oldValue);
564.                                    }
565.                                } else if (!poRevert) { // The user selected another PO
    Number and did not selected a Product beforehand.
566.                                    model.clearProducts();
567.                                    try { // Populate cmbProducts
568.
    model.productsProperty().setAll(DBService.getInstance().getPurchaseOrderProducts(newValue));
569.                                        poConnectionError = false;
570.                                    } catch (SQLException e) {
571.                                        Alert a = new Alert(Alert.AlertType.ERROR);
572.                                        a.setContentText(
573.                                            "Error while connecting to SQL Server
    database!" + System.lineSeparator() +
574.                                            e.getLocalizedMessage() +
    System.lineSeparator() +
```

```
575.                                                 "Please select the value again to try again."
576.                                             );
577.
    a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
578.                                             a.showAndWait();
579.                                             poConnectionError = true;
580.                                         }
581.                                     } else // Let the change go through and reset poRevert.
582.                                         poRevert = false;
583.                                 }
584.                             } else if (newValue != null) { // They chose a PO number and
    oldValue was null.
585.                                 try { // Populate cmbProducts
586.
    model.productsProperty().setAll(DBService.getInstance().getPurchaseOrderProducts(newValue));
587.                                     cmbProduct.setDisable(false);
588.                                     poConnectionError = false;
589.                                 } catch (SQLException e) {
590.                                     Alert a = new Alert(Alert.AlertType.ERROR);
591.                                     a.setContentText(
592.                                         "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
593.                                         e.getLocalizedMessage() + System.lineSeparator() +
594.                                         "Please select the value again to try again."
595.                                     );
596.                                     a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
597.                                     a.showAndWait();
598.                                     poConnectionError = true;
599.                                 }
600.                             }
601.                         }
602.                     }
603.             );
604.
605.             cmbCreditReplaceRepair.getSelectionModel().selectedItemProperty().addListener(
606.                 new ChangeListener<String>() {
607.                     /** changed checks the new and previous value to decide whether to clear
    and
608.                      *  disable the fields under Replacement Information, or re-enable them,
609.                      *  depending on whether the user selects the "Credit" option.
610.                      * @param observableValue The {@link ObservableValue} variable that
    changed.
611.                      * @param oldValue        The old {@link String} value of the {@link
    ObservableValue}.
612.                      * @param newValue        The new {@link String} value of the {@link
    ObservableValue}.
613.                      */
614.                     @Override
615.                     public void changed(ObservableValue<? extends String> observableValue,
    String oldValue, String newValue) {
616.                         // First handle the user selecting "Credit".
617.                         if (newValue != null) // null is passed in when
    getSelectionModel().clearSelection() is called.
618.                             if (newValue.equals("Credit")) {
619.                                 // Check any of the Replacement Information fields are
    populated.
620.                                 if (
621.                                    (!model.getReplacementTrackingNumber().isEmpty() &&
    !model.getReplacementTrackingNumber().isBlank()) ||
622.                                     model.getReplacementShipDate() != null ||
623.                                     model.getShipReplacementRepair()
624.                                 ) {
625.                                     // Ask the user if they want to continue with clearing and
    disabling the fields.
```

```
626.                                 ButtonType confirm = new ButtonType("Confirm",
    ButtonBar.ButtonData.OK_DONE); // Change the button context from OK to Confirm
627.                                 Alert a = new Alert(
628.                                     Alert.AlertType.CONFIRMATION,
629.                                     "Selecting \"Credit\" will clear all data entered
    under the " +
630.                                        "\"Replacement Information\" section and disable
    those controls " +
631.                                        "because the customer will not be receiving a
    replacement or repair when " +
632.                                        "they are receiving credit. Do you want to
    continue?",
633.                                     confirm,
634.                                     ButtonType.CANCEL
635.                                 );
636.                                 a.setTitle("Clearing Replacement Information
    Verification");
637.                                 a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
638.                                 Optional<ButtonType> result = a.showAndWait();
639.
640.                                 // Clear and disable the controls under Replacement
    Information.
641.                                 if (result.isPresent() && result.get() == confirm) {
642.                                     model.setReplacementTrackingNumber("");
643.                                     txtReplaceRepairTracking.setDisable(true);
644.                                     model.setReplacementShipDate(null);
645.                                     dtpkrReplaceRepairShipDate.setDisable(true);
646.                                     model.setShipReplacementRepair(false);
647.                                     cbShipIndicator.setDisable(true);
648.                                 } else // Set value back to empty using
    Platform.runLater() or to its previous value.
649.                                     if (oldValue == null)
650.                                         Platform.runLater(() ->
    cmbCreditReplaceRepair.getSelectionModel().clearSelection());
651.                                     else
652.                                         model.setSelectedCreditReplaceRepair(oldValue);
653.                             } else { // Disable the controls.
654.                                 txtReplaceRepairTracking.setDisable(true);
655.                                 dtpkrReplaceRepairShipDate.setDisable(true);
656.                                 cbShipIndicator.setDisable(true);
657.                             }
658.                         } else { // Re-enable the controls.
659.                             txtReplaceRepairTracking.setDisable(false);
660.                             dtpkrReplaceRepairShipDate.setDisable(false);
661.                             cbShipIndicator.setDisable(false);
662.                         }
663.                     }
664.                 }
665.             );
666.         }
667.
668.         /**
669.          * cmbCustomerNameOnHidden checks, after the dropdown menu closes, whether the user
    had experienced a
670.          * customerConnectionError, and if so, attempts the query again to populate Business
    Name.
671.          *
672.          * @param event The {@link Event} passed to the method.
673.          */
674.         @FXML
675.         public void cmbCustomerNameOnHidden(Event event) {
676.             if (customerConnectionError)
677.                 try { // Populate cmbBusinessName
678.                     model.setSelectedBusinessName(null);
```

```
679.
    model.businessNamesProperty().setAll(DBService.getInstance().getCustomerBusinessNames(model.
    getSelectedCustomerName()));
680.                    cmbBusinessName.setDisable(false);
681.                    customerConnectionError = false;
682.                } catch (SQLException e) {
683.                    Alert a = new Alert(Alert.AlertType.ERROR);
684.                    a.setContentText(
685.                            "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
686.                                    e.getLocalizedMessage() + System.lineSeparator() +
687.                                    "Please select the value again to try again."
688.                    );
689.                    a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
690.                    a.showAndWait();
691.                    customerConnectionError = true;
692.                }
693.        }
694.
695.        /**
696.         * cmbBusinessNameOnHidden checks, after the dropdown menu closes, whether the user
    had experienced a
697.         * businessConnectionError, and if so, attempts the query again to populate PO Number.
698.         *
699.         * @param event The {@link Event} passed to the method.
700.         */
701.        @FXML
702.        public void cmbBusinessNameOnHidden(Event event) {
703.            if (businessConnectionError) // Attempt the connection again if the user got an
    error last time.
704.                try { // Populate cmbPoNumber
705.                    model.setSelectedPONumber(null);
706.
    model.poNumbersProperty().setAll(DBService.getInstance().getCustomerAddressPONumbers(model.g
    etSelectedBusinessName()));
707.                    cmbPoNumber.setDisable(false);
708.                    businessConnectionError = false;
709.                } catch (SQLException e) {
710.                    Alert a = new Alert(Alert.AlertType.ERROR);
711.                    a.setContentText(
712.                            "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
713.                                    e.getLocalizedMessage() + System.lineSeparator() +
714.                                    "Please select the value again to try again."
715.                    );
716.                    a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
717.                    a.showAndWait();
718.                    businessConnectionError = true;
719.                }
720.        }
721.
722.        /**
723.         * cmbPONumberOnHidden checks, after the dropdown menu closes, whether the user had
    experienced a
724.         * poConnectionError, and if so, attempts the query again to populate Product.
725.         *
726.         * @param event The {@link Event} passed to the method.
727.         */
728.        @FXML
729.        public void cmbPONumberOnHidden(Event event) {
730.            if (poConnectionError) // Attempt the connection again if the user got an error
    last time.
731.                try { // Populate cmbProducts
732.                    model.setSelectedProduct(null);
733.                    model.clearProducts();
```

```
734.
    model.productsProperty().setAll(DBService.getInstance().getPurchaseOrderProducts(model.getSe
    lectedPONumber()));
735.                    cmbProduct.setDisable(false);
736.                    poConnectionError = false;
737.                } catch (SQLException e) {
738.                    Alert a = new Alert(Alert.AlertType.ERROR);
739.                    a.setContentText(
740.                            "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
741.                                    e.getLocalizedMessage() + System.lineSeparator() +
742.                                    "Please select the value again to try again."
743.                    );
744.                    a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
745.                    a.showAndWait();
746.                    poConnectionError = true;
747.                }
748.        }
749.
750.
751.        /**
752.         * btnCancelOnAction closes the screen without saving.
753.         * (Requirement 4.10.1, 4.10.2)
754.         * @param event The {@link ActionEvent} that is passed to the method.
755.         */
756.        @FXML
757.        protected void btnCancelOnAction(ActionEvent event) {
758.            // Ask the user if they want to cancel and close the form.
759.            ButtonType confirm = new ButtonType("Confirm", ButtonBar.ButtonData.OK_DONE); //
    Change the button context from OK to Confirm
760.            Alert a = new Alert(
761.                    Alert.AlertType.CONFIRMATION,
762.                    "All data input will be lost and the form will be closed! Would you like
    to continue?",
763.                    confirm,
764.                    ButtonType.CANCEL
765.            );
766.            a.setTitle("Clear and Close Form without Saving");
767.            a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
768.            Optional<ButtonType> result = a.showAndWait();
769.
770.            // Clear and close the form.
771.            if (result.isPresent() && result.get() == confirm) {
772.                model.setSelectedCustomerName(null);
773.                model.setSelectedBusinessName(null);
774.                model.clearBusinessNames();
775.                model.setSelectedPONumber(null);
776.                model.clearPONumbers();
777.                model.setSelectedProduct(null);
778.                model.setShippingAddress(CustomerAddress.EMPTY_ADDRESS);
779.                model.setSelectedReturnReasonCode("");
780.                model.setSelectedCreditReplaceRepair("");
781.                model.setSelectedRMAStatus("");
782.                model.setAdditionalInfo("");
783.                model.setSelectedProduct(null);
784.                model.clearProducts();
785.                model.setReturnQuantity(0);
786.                model.setReturnLabelTracker("");
787.                model.setInitialEvaluation("");
788.                model.setSelectedDisposition("");
789.                model.setDispositionNotes("");
790.                model.setReplacementTrackingNumber("");
791.                model.setReplacementShipDate(null);
792.                model.setShipReplacementRepair(false);
793.                btnCancel.getScene().getWindow().hide();
```

```
794.          }
795.      }
796.
797.      /**
798.       * btnSaveOnAction Saves the RMA and all the data into the database.
799.       * (Requirement 4.6, 4.10.1, 4.10.2)
800.       * @param event The {@link ActionEvent} that is passed to the method.
801.       */
802.      @FXML
803.      protected void btnSaveOnAction(ActionEvent event) {
804.          Alert miss = new Alert(Alert.AlertType.INFORMATION,
805.                  "One or more of the required fields (with an asterisk * by the name) are
    missing input. Please complete before continuing."
806.          );
807.          miss.setTitle("Required Field Missing!");
808.          miss.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
809.
810.          Alert save = new Alert(Alert.AlertType.INFORMATION, "New RMA created and saved.");
811.          save.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
812.          save.setTitle("RMA Save Confirmed");
813.
814.          // Check if all required fields are filled in.
815.          if (!model.getSelectedCustomerName().isEmpty() &&
816.               model.getSelectedBusinessName() != null &&
817.              !model.getSelectedPONumber().isEmpty() &&
818.              !model.getSelectedReturnReasonCode().isEmpty() &&
819.              !model.getSelectedCreditReplaceRepair().isEmpty() &&
820.              !model.getSelectedRMAStatus().isEmpty() &&
821.               model.getSelectedProduct() != null &&
822.               model.getReturnQuantity() > 0
823.          ) {
824.              // Ask the user if they want to save the RMA before processing.
825.              ButtonType confirm = new ButtonType("Confirm", ButtonBar.ButtonData.OK_DONE);
    // Change the button context from OK to Confirm
826.              Alert a = new Alert(
827.                      Alert.AlertType.CONFIRMATION,
828.                      "Would you like to save the RMA?",
829.                      confirm,
830.                      ButtonType.CANCEL
831.              );
832.              a.setTitle("Save RMA Verification");
833.              a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
834.              Optional<ButtonType> result = a.showAndWait();
835.
836.              if (result.isPresent() && result.get() == confirm)
837.                  try {
838.                      // Save RMA into the database.
839.                      DBService dbs = DBService.getInstance();
840.                      model.setRmaId(dbs.createRMA(model.getSelectedRMAStatus()));
841.                      dbs.createRMADetails(
842.                          model.getRmaId(),
843.                          model.getSelectedReturnReasonCode().substring(0,
    model.getSelectedReturnReasonCode().indexOf(" ")),
844.                          model.getSelectedCreditReplaceRepair(),
845.                          model.getSelectedProduct().getPurchaseOrderProductId(),
846.                          model.getReturnQuantity(),
847.                          model.getReturnLabelTracker(),
848.                          model.getAdditionalInfo(),
849.                          model.getSelectedPONumber(),
850.                          model.getInitialEvaluation(),
851.                          "", // engineeringEvaluation
852.                          model.getSelectedDisposition(),
853.                          model.getDispositionNotes(),
854.                          model.getReplacementTrackingNumber(),
855.                          model.getReplacementShipDate(),
```

```
856.                         model.getShipReplacementRepair()
857.                     );
858.                     lblOwnerValue.setText(dbs.getUser());
859.                     save.showAndWait();
860.                 } catch (SQLException e) {
861.                     Alert error = new Alert(Alert.AlertType.ERROR);
862.                     error.setContentText("Error while connecting to SQL Server database!"
    + System.lineSeparator() + e.getLocalizedMessage());
863.                     error.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
864.                     error.show();
865.                 }
866.             } else
867.                 miss.showAndWait();
868.         }
869.
870.         /**
871.          * btnSaveCloseAction Saves the RMA and all the data into the database and closes the
    form.
872.          * (Requirement 4.6, 4.10.1, 4.10.2)
873.          * @param event The {@link ActionEvent} that is passed to the method.
874.          */
875.         @FXML
876.         protected void btnSaveCloseAction(ActionEvent event) {
877.             Alert miss = new Alert(Alert.AlertType.INFORMATION,
878.                     "One or more of the required fields (with an asterisk * by the name) are
    missing input. Please complete before continuing."
879.             );
880.             miss.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
881.             miss.setTitle("Required Field Missing!");
882.
883.             Alert save = new Alert(Alert.AlertType.INFORMATION, "New RMA created and saved.");
884.             save.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
885.             save.setTitle("RMA Save Confirmed");
886.
887.             if (!model.getSelectedCustomerName().isEmpty() &&
888.                     model.getSelectedBusinessName() != null &&
889.                     !model.getSelectedPONumber().isEmpty() &&
890.                     !model.getSelectedReturnReasonCode().isEmpty() &&
891.                     !model.getSelectedCreditReplaceRepair().isEmpty() &&
892.                     !model.getSelectedRMAStatus().isEmpty() &&
893.                     model.getSelectedProduct() != null &&
894.                     model.getReturnQuantity() > 0
895.             ) {
896.                 // Ask the user if they want to save and close the RMA before processing.
897.                 ButtonType confirm = new ButtonType("Confirm", ButtonBar.ButtonData.OK_DONE);
    // Change the button context from OK to Confirm
898.                 Alert a = new Alert(
899.                         Alert.AlertType.CONFIRMATION,
900.                         "Would you like to save and close the RMA?",
901.                         confirm,
902.                         ButtonType.CANCEL
903.                 );
904.                 a.setTitle("Save and Close RMA Verification");
905.                 a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
906.                 Optional<ButtonType> result = a.showAndWait();
907.
908.                 if (result.isPresent() && result.get() == confirm)
909.                     try {
910.                         // Save RMA into the database.
911.                         DBService dbs = DBService.getInstance();
912.                         model.setRmaId(dbs.createRMA(model.getSelectedRMAStatus()));
913.                         dbs.createRMADetails(
914.                             model.getRmaId(),
915.                             model.getSelectedReturnReasonCode().substring(0,
    model.getSelectedReturnReasonCode().indexOf(" ")),
```

```
916.                            model.getSelectedCreditReplaceRepair(),
917.                            model.getSelectedProduct().getPurchaseOrderProductId(),
918.                            model.getReturnQuantity(),
919.                            model.getReturnLabelTracker(),
920.                            model.getAdditionalInfo(),
921.                            model.getSelectedPONumber(),
922.                            model.getInitialEvaluation(),
923.                            "", // engineeringEvaluation
924.                            model.getSelectedDisposition(),
925.                            model.getDispositionNotes(),
926.                            model.getReplacementTrackingNumber(),
927.                            model.getReplacementShipDate(),
928.                            model.getShipReplacementRepair()
929.                        );
930.                        lblOwnerValue.setText(dbs.getUser());
931.                        save.showAndWait();
932.                        btnSaveClose.getScene().getWindow().hide();
933.                    } catch(SQLException e) {
934.                        Alert error = new Alert(Alert.AlertType.ERROR);
935.                        error.setContentText("Error while connecting to SQL Server database!"
    + System.lineSeparator() + e.getLocalizedMessage());
936.                        error.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
937.                        error.show();
938.                    }
939.            } else
940.                miss.showAndWait();
941.        }
942.
943.        /** getInitialRMAOptions fetches the Customer names, Return Reason Codes, RMA
    Statuses, and Dispositions from the
944.         *  database.
945.         *  (Requirement 4.5.1)
946.         * @throws SQLException If there is an issue connecting to the database.
947.         */
948.        public void getInitialRMAOptions() throws SQLException {
949.            // First create an instance of DBService.
950.            DBService dbs = DBService.getInstance();
951.
952.            // Now populate dropdowns.
953.            model.customerNamesProperty().setAll(dbs.getCustomerNames());
954.
    model.returnReasonCodesProperty().setAll(model.convertHashMapToArrayList(dbs.getReturnReason
    Codes()));
955.            ArrayList<String> list = dbs.getRMAStatuses();
956.            list.remove("Closed");
957.            model.rmaStatusesProperty().setAll(list);
958.            model.dispositionsProperty().setAll(dbs.getDispositions());
959.        }
960.    }
961.
```

RMAFormModel.java

```
1.  package RMA;
2.
3.  import javafx.beans.property.*;
4.  import javafx.collections.FXCollections;
5.  import javafx.collections.ObservableList;
6.
7.  import java.time.LocalDate;
8.  import java.time.LocalDateTime;
9.  import java.time.format.DateTimeFormatter;
10. import java.util.ArrayList;
11. import java.util.HashMap;
12.
13. /** RMAFormModel contains the fields and methods that make up the RMAForm model.
14.  */
15. public class RMAFormModel {
16.     /** rmaId contains the RMA ID to identify the RMA.
17.      */
18.     private StringProperty rmaId;
19.     /** created contains the date and time that the RMA was created.
20.      */
21.     private ObjectProperty<LocalDateTime> created;
22.     /** createdBy holds the username of the RMA creator.
23.      */
24.     private StringProperty createdBy;
25.     /** lastModified contains the date and time the RMA was last modified.
26.      */
27.     private ObjectProperty<LocalDateTime> lastModified;
28.     /** lastModifiedBy contains the username of the last person to modify the RMA record.
29.      */
30.     private StringProperty lastModifiedBy;
31.     /** selectedCustomerName contains the selected customer name.
32.      */
33.     private StringProperty selectedCustomerName;
34.     /** customerNames contains the {@link ObservableList} of {@link String} customer names.
35.      */
36.     private ObservableList<String> customerNames;
37.     /** selectedBusinessName contains the selected business name CustomerAddress of the
    selected customer name.
38.      */
39.     private ObjectProperty<CustomerAddress> selectedBusinessName;
40.     /** businessNames contains the {@link ObservableList} of {@link CustomerAddress}
    addresses for the selected
41.      *  customer name, with their showBusinessName set to true.
42.      */
43.     private ObservableList<CustomerAddress> businessNames;
44.     /** shippingAddress contains the the business's address.
45.      */
46.     private ObjectProperty<CustomerAddress> shippingAddress;
47.     /** selectedRMAStatus contains the selected RMA Status for the RMA.
48.      */
49.     private StringProperty selectedRMAStatus;
50.     /** rmaStatuses contains the {@link ObservableList} of {@link String} RMA statuses for
    the RMA.
51.      */
52.     private ObservableList<String> rmaStatuses;
53.     /** selectedPONumber contains the selected PO number from the available PO numbers for
    the selected business name.
54.      */
55.     private StringProperty selectedPONumber;
56.     /** poNumbers contains the {@link ObservableList} of {@link String}s containing the
    available PO numbers for the
```

```
57.      *  given business name.
58.      */
59.     private ObservableList<String> poNumbers;
60.     /** selectedReturnReasonCode contains the selected Return Reason Code.
61.      */
62.     private StringProperty selectedReturnReasonCode;
63.     /** returnReasonCodes contains the {@link ObservableList} of {@link String} Return
    Reason Codes.
64.      */
65.     private ObservableList<String> returnReasonCodes;
66.     /** selectedCreditReplaceRepair contains the selected Credit, Replace, or Repair option.
67.      */
68.     private StringProperty selectedCreditReplaceRepair;
69.     /** creditReplaceRepair contains the {@link ObservableList} of {@link String}s,
    specifically Credit, Replace, and
70.      *  Repair.
71.      */
72.     private ObservableList<String> creditReplaceRepair;
73.     /** additionalInfo contains any additional user-provided information about the RMA
    return.
74.      */
75.     private StringProperty additionalInfo;
76.     /** selectedProduct contains the selected PurchaseOrderProduct.
77.      */
78.     private ObjectProperty<PurchaseOrderProduct> selectedProduct;
79.     /** products contains the {@link ObservableList} of available PurchaseOrderProducts for
    the selected PO number.
80.      */
81.     private ObservableList<PurchaseOrderProduct> products;
82.     /** returnQuantity contains the number of products being returned.
83.      */
84.     private IntegerProperty returnQuantity;
85.     /** returnLabelTracker contains the customer-provided return label tracking number for
    the given RMA return.
86.      */
87.     private StringProperty returnLabelTracker;
88.     /** initialEvaluation contains the initial evaluation of the returned product(s) by an
    Analyst.
89.      */
90.     private StringProperty initialEvaluation;
91.     /** selectedDisposition contains the selected disposition description.
92.      */
93.     private StringProperty selectedDisposition;
94.     /** dispositions contains the {@link ObservableList} of {@link String}s containing
    disposition descriptions.
95.      */
96.     private ObservableList<String> dispositions;
97.     /** dispositionNotes contains any additional Analyst-provided notes about the chosen
    disposition.
98.      */
99.     private StringProperty dispositionNotes;
100.      /** replacementTrackingNumber contains the tracking number used to ship the
    replacement or repair back to the
101.       *  customer.
102.       */
103.      private StringProperty replacementTrackingNumber;
104.      /** replacementShipDate contains shipping date of the our replacement or return.
105.       */
106.      private ObjectProperty<LocalDate> replacementShipDate;
107.      /** shipReplacementRepair is used to indicate that we are shipping the replacement or
    repair.
108.       */
109.      private BooleanProperty shipReplacementRepair;
110.
111.
```

```
112.      /** Default constructor that initializes fields to empty values (besides
    creditReplaceRepair,
113.       *  which contains fixed values).
114.       */
115.      public RMAFormModel() {
116.          // Initialize properties
117.          rmaId = new SimpleStringProperty();
118.          created = new SimpleObjectProperty<LocalDateTime>();
119.          createdBy = new SimpleStringProperty();
120.          lastModified = new SimpleObjectProperty<LocalDateTime>();
121.          lastModifiedBy = new SimpleStringProperty();
122.          selectedCustomerName = new SimpleStringProperty();
123.          customerNames = FXCollections.observableArrayList();
124.          selectedBusinessName = new SimpleObjectProperty<CustomerAddress>();
125.          businessNames = FXCollections.observableArrayList();
126.          shippingAddress = new
    SimpleObjectProperty<CustomerAddress>(CustomerAddress.EMPTY_ADDRESS);
127.          selectedRMAStatus = new SimpleStringProperty();
128.          rmaStatuses = FXCollections.observableArrayList();
129.          selectedPONumber = new SimpleStringProperty();
130.          poNumbers = FXCollections.observableArrayList();
131.          selectedReturnReasonCode = new SimpleStringProperty();
132.          returnReasonCodes = FXCollections.observableArrayList();
133.          selectedCreditReplaceRepair = new SimpleStringProperty();
134.          creditReplaceRepair = FXCollections.observableArrayList("Credit", "Replace",
    "Repair");
135.          additionalInfo = new SimpleStringProperty();
136.          selectedProduct = new SimpleObjectProperty<PurchaseOrderProduct>();
137.          products = FXCollections.observableArrayList();
138.          returnQuantity = new SimpleIntegerProperty();
139.          returnLabelTracker = new SimpleStringProperty();
140.          initialEvaluation = new SimpleStringProperty();
141.          selectedDisposition = new SimpleStringProperty();
142.          dispositions = FXCollections.observableArrayList();
143.          dispositionNotes = new SimpleStringProperty();
144.          replacementTrackingNumber = new SimpleStringProperty();
145.          replacementShipDate = new SimpleObjectProperty<>();
146.          shipReplacementRepair = new SimpleBooleanProperty();
147.      }
148.
149.      /** getRmaId returns the {@link String} RMA ID that was created on save.
150.       * @return The {@link String} RMA ID that was entered in the database.
151.       */
152.      public String getRmaId() {return rmaId.getValueSafe();}
153.
154.      /** setRmaId sets the rmaId to the provided newRMAId.
155.       * @param newRMAId {@link String} value of the ID created on save.
156.       */
157.      public void setRmaId(String newRMAId) {rmaId.setValue(newRMAId);}
158.
159.      /** rmaIDProperty returns the {@link StringProperty} field containing the RMA ID.
160.       * @return The {@link StringProperty} rmaId field.
161.       */
162.      public StringProperty rmaIDProperty() {return rmaId;}
163.
164.      /** getSelectedCustomerName returns the {@link String} selected customer name stored
    in the selectedCustomerName field.
165.       * @return A {@link String} containing the customer name selected by the user.
166.       */
167.      public String getSelectedCustomerName() {return selectedCustomerName.getValueSafe();}
168.
169.      /** setSelectedCustomerName sets the selectedCustomerName field value to the passed-in
    newSelectedCustomerName {@link String}.
170.       * @param newSelectedCustomerName The new {@link String} customer name selected by the
    user.
```

```
171.       */
172.      public void setSelectedCustomerName(String newSelectedCustomerName)
    {selectedCustomerName.setValue(newSelectedCustomerName);}
173.
174.      /** getSelectedCustomerNameProperty returns the {@link StringProperty} containing the
    selected customer name.
175.       * @return The {@link StringProperty} selectedCustomerName field.
176.       */
177.      public StringProperty selectedCustomerNameProperty() {return selectedCustomerName;}
178.
179.      /** customerNamesProperty returns the {@link ObservableList} of {@link String}
    customer names.
180.       * @return The {@link ObservableList} customerNames field.
181.       */
182.      public ObservableList<String> customerNamesProperty() {return customerNames;}
183.
184.      /** getSelectedBusinessName returns the selected {@link String} business name stored
    in selectedBusinessName.
185.       * @return A {@link String} containing the business name selected by the user.
186.       */
187.      public CustomerAddress getSelectedBusinessName() {return selectedBusinessName.get();}
188.
189.      /** setSelectedBusinessName sets the selectedBusinessName field value to the passed-in
    newSelectedBusinessName {@link String}.
190.       * @param newSelectedBusinessName The new {@link String} business name selected by the
    user.
191.       */
192.      public void setSelectedBusinessName(CustomerAddress newSelectedBusinessName)
    {selectedBusinessName.setValue(newSelectedBusinessName);}
193.
194.      /** selectedBusinessNameProperty returns the {@link StringProperty} containing the
    selected business name.
195.       * @return The {@link StringProperty} selectedBusinessName field.
196.       */
197.      public ObjectProperty<CustomerAddress> selectedBusinessNameProperty() {return
    selectedBusinessName;}
198.
199.      /** clearBusinessNames clears the {@link ObservableList} containing the list of
    business names for the given
200.       *  customer name.
201.       */
202.      public void clearBusinessNames() {businessNames.clear();}
203.
204.      /** businessNamesProperty returns the {@link ObservableList} of {@link
    CustomerAddress} addresses
205.       *  with their showBusinessName set to true.
206.       * @return The {@link ObservableList} businessNames field.
207.       */
208.      public ObservableList<CustomerAddress> businessNamesProperty() {return businessNames;}
209.
210.      /** getCreated returns the {@link LocalDateTime} stored in created.
211.       * @return The {@link LocalDateTime} stored in created.
212.       */
213.      public LocalDateTime getCreated() {return created.get();}
214.
215.      /** setCreated sets the {@link LocalDateTime} date and time stored in the created
    field to the
216.       *  new user-entered newCreated {@link LocalDateTime}.
217.       * @param newCreated The new {@link LocalDateTime} date and time of when the RMA was
    created
218.       */
219.      public void setCreated(LocalDateTime newCreated) {created.set(newCreated);}
220.
221.      /** createdProperty returns the {@link ObjectProperty} containing the
222.       *  creation {@link LocalDateTime} date and time of this RMA request.
```

```
223.        * @return The {@link ObjectProperty} created field.
224.        */
225.       public ObjectProperty<LocalDateTime> createdProperty() {return created;}
226.
227.        /** getCreatedBy returns the username {@link String} that created the RMA request.
228.        * @return The {@link String} stored in the createdBy field.
229.        */
230.       public String getCreatedBy() {return createdBy.getValueSafe();}
231.
232.       /** setCreatedBy sets the username that created this RMA request to the passed-in
    newCreatedBy {@link String} value.
233.        * @param newCreatedBy The new {@link String} value of the name of employee that
    created the RMA.
234.        */
235.       public void setCreatedBy(String newCreatedBy) {createdBy.setValue(newCreatedBy);}
236.
237.       /** createdByProperty returns the {@link StringProperty} createdBy field.
238.        * @return The {@link StringProperty} createdBy field.
239.        */
240.       public StringProperty createdByProperty() {return createdBy;}
241.
242.       /** getLastModified returns the {@link LocalDateTime} stored in the getLastModified
    field.
243.        * @return The {@link LocalDateTime} stored in lastModified field.
244.        */
245.       public LocalDateTime getLastModified() {return lastModified.get();}
246.
247.       /** setLastModified sets the {@link LocalDateTime} date and time value stored in
    lastModified to the new
248.        *  provided newLastModified {@link LocalDateTime} value.
249.        * @param newLastModified The new {@link LocalDateTime} value of when RMA was last
    modified.
250.        */
251.       public void setLastModified(LocalDateTime newLastModified)
    {lastModified.set(newLastModified);}
252.
253.       /** lastModifiedProperty returns the {@link ObjectProperty} containing the
254.        *  lastModified {@link LocalDateTime} date and time of this RMA.
255.        * @return The {@link ObjectProperty} lastModified field.
256.        */
257.       public ObjectProperty<LocalDateTime> lastModifiedProperty() {return lastModified;}
258.
259.       /** getLastModifiedBy returns the name {@link String} employee name stored in the
    lastModifiedBy field.
260.        * @return A {@link String} containing the name stored in lastModifiedBy.
261.        */
262.       public String getLastModifiedBy() {return lastModifiedBy.getValueSafe();}
263.
264.       /** setLastModifiedBy sets the {@link String} name of employee that modified the RMA
    record to the passed-in
265.        *  {@link String} newLastModifiedBy value.
266.        * @param newLastModifiedBy The {@link String} username that last modified the RMA.
267.        */
268.       public void setLastModifiedBy(String newLastModifiedBy)
    {lastModifiedBy.setValue(newLastModifiedBy);}
269.
270.       /** lastModifiedByProperty returns the {@link StringProperty} lastModifiedBy field.
271.        * @return The {@link StringProperty} lastModifiedBy field.
272.        */
273.       public StringProperty lastModifiedByProperty() {return lastModifiedBy;}
274.
275.       /** getSelectedRMAStatus returns the {@link String} RMA status stored in
    selectedRMAStatus.
276.        * @return The {@link String} selectedRMAStatus value.
277.        */
```

```
278.      public String getSelectedRMAStatus() {return selectedRMAStatus.getValueSafe();}
279.
280.      /** setSelectedRMAStatus sets the RMA status stored in selectedRMAStatus to the
    passed-in {@link String}
281.       *  newRMAStatus value selected by the user.
282.       * @param newRMAStatus The new {@link String} RMA Status selected by the user.
283.       */
284.      public void setSelectedRMAStatus(String newRMAStatus)
    {selectedRMAStatus.setValue(newRMAStatus);}
285.
286.      /** selectedRMAStatusProperty returns the {@link StringProperty} selectedRMAStatus
    field.
287.       * @return The {@link StringProperty} selectedRMAStatus field.
288.       */
289.      public StringProperty selectedRMAStatusProperty() {return selectedRMAStatus;}
290.
291.      /** rmaStatusesProperty returns the {@link ObservableList} of {@link String} RMA
    statuses.
292.       * @return The {@link ObservableList} rmaStatuses field.
293.       */
294.      public ObservableList<String> rmaStatusesProperty() {return rmaStatuses;}
295.
296.      /** getSelectedPONumber returns the {@link String} Purchase Order number selected by
    the user.
297.       * @return The {@link String} stored in the selectedPONumber field.
298.       */
299.      public String getSelectedPONumber() {return selectedPONumber.getValueSafe();}
300.
301.      /** setSelectedPONumber sets the Purchase Order number stored in selectedPONumber to
    the passed-in
302.       *  {@link String} newSelectedPONumber selected by the user.
303.       * @param newSelectedPONumber The new {@link String} PO number selected by the user.
304.       */
305.      public void setSelectedPONumber(String newSelectedPONumber)
    {selectedPONumber.setValue(newSelectedPONumber);}
306.
307.      /** selectedPONumberProperty returns the {@link StringProperty} selectedPONumber
    field.
308.       * @return The {@link StringProperty} selectedPONumber field.
309.       */
310.      public StringProperty selectedPONumberProperty() {return selectedPONumber;}
311.
312.      /** clearPONumbers clears the {@link ObservableList} poNumbers field.
313.       */
314.      public void clearPONumbers() {poNumbers.clear();}
315.
316.      /** poNumbersProperty returns the {@link ObservableList} poNumbers field containing
    the list of Purchase Order
317.       *  numbers associated with a given business name.
318.       * @return The {@link ObservableList} poNumbers field.
319.       */
320.      public ObservableList<String> poNumbersProperty() {return poNumbers;}
321.
322.      /** getSelectedReturnReasonCode returns the {@link String} Return Reason Code selected
    by the user.
323.       * @return The {@link String} Return Reason Code selected by the user.
324.       */
325.      public String getSelectedReturnReasonCode() {return
    selectedReturnReasonCode.getValueSafe();}
326.
327.      /** setSelectedReturnReasonCode sets the {@link String} Return Reason Code selected by
    the user to
328.       *  selectedReturnReasonCode using the passed-in parameter.
329.       *  newReturnReasonCode {@link String} value.
```

```
330.       * @param newReturnReasonCode The new {@link String} Return Reason Code to store in
    selectedReturnReasonCode.
331.       */
332.      public void setSelectedReturnReasonCode(String newReturnReasonCode)
    {selectedReturnReasonCode.setValue(newReturnReasonCode);}
333.
334.      /** selectedReturnReasonCodeProperty returns the {@link StringProperty}
    selectedReturnReasonCode field.
335.       * @return The {@link StringProperty} selectedReturnReasonCode field.
336.       */
337.      public StringProperty selectedReturnReasonCodeProperty() {return
    selectedReturnReasonCode;}
338.
339.      /** returnReasonCodesProperty returns the {@link ObservableList} of {@link String}
    return reason code values
340.       *  stored in the returnReasonCodes field.
341.       * @return The {@link ObservableList} returnReasonCodes field.
342.       */
343.      public ObservableList<String> returnReasonCodesProperty() {return returnReasonCodes;}
344.
345.      /** getSelectedCreditReplaceRepair returns the {@link String} Credit, Replace, or
    Repair value selected by the user.
346.       * @return The {@link String} Credit, Replace, or Repair value selected by the user.
347.       */
348.      public String getSelectedCreditReplaceRepair() {return
    selectedCreditReplaceRepair.getValueSafe();}
349.
350.      /** setSelectedCreditReplaceRepair sets the new {@link String} creditReplaceRepair
    field value to the
351.       *  passed-in {@link String} newCreditReplaceRepair parameter value.
352.       * @param newCreditReplaceRepair The new {@link String} value to set.
353.       */
354.      public void setSelectedCreditReplaceRepair(String newCreditReplaceRepair)
    {selectedCreditReplaceRepair.setValue(newCreditReplaceRepair);}
355.
356.      /** getSelectedCreditReplaceRepairProperty returns the {@link StringProperty}
357.       * @return selectedCreditReplaceRepair value.
358.       */
359.      public StringProperty selectedCreditReplaceRepairProperty() {return
    selectedCreditReplaceRepair;}
360.
361.      /** creditReplaceRepairProperty returns the {@link ObservableList} of {@link String}
    Credit, Replace, and Repair values.
362.       * @return The {@link ObservableList} creditReplaceRepair field.
363.       */
364.      public ObservableList<String> creditReplaceRepairProperty() {return
    creditReplaceRepair;}
365.
366.      /** getShippingAddress returns the {@link CustomerAddress} stored in the
    shippingAddress field.
367.       * @return The {@link CustomerAddress} stored in shippingAddress.
368.       */
369.      public CustomerAddress getShippingAddress() {return shippingAddress.get();}
370.
371.      /** setShippingAddress sets the stored {@link CustomerAddress} shipping address value
    stored in shippingAddress to
372.       *  the passed-in newShippingAddress value.
373.       * @param newShippingAddress The new {@link CustomerAddress} to store in
    shippingAddress.
374.       */
375.      public void setShippingAddress(CustomerAddress newShippingAddress)
    {shippingAddress.set(newShippingAddress);}
376.
377.      /** shippingAddressProperty returns the {@link ObjectProperty} shippingAddress field
    containing a
```

```
378.       *  {@link CustomerAddress}.
379.       * @return The {@link ObjectProperty} of {@link CustomerAddress} shippingAddress
    field.
380.       */
381.      public ObjectProperty<CustomerAddress> shippingAddressProperty() {return
    shippingAddress;}
382.
383.      /** getAdditionalInfo returns the {@link String} stored in the additionalInfo field.
384.       * @return The {@link String} stored in the additionalInfo field.
385.       */
386.      public String getAdditionalInfo() {return additionalInfo.getValueSafe();}
387.
388.      /** setAdditionalInfo sets the passed-in {@link String} value to the additionalInfo
    field.
389.       * @param newAdditionalInfo The new {@link String} additional info value to store in
    additionalInfo.
390.       */
391.      public void setAdditionalInfo(String newAdditionalInfo)
    {additionalInfo.setValue(newAdditionalInfo);}
392.
393.      /** additionalInfoProperty returns the {@link StringProperty} additionalInfo field.
394.       * @return The {@link StringProperty} additionalInfo field.
395.       */
396.      public StringProperty additionalInfoProperty() {return additionalInfo;}
397.
398.      /** getSelectedProduct returns the {@link PurchaseOrderProduct} stored in the
    selectedProduct field.
399.       * @return The {@link PurchaseOrderProduct} value stored in selectedProduct.
400.       */
401.      public PurchaseOrderProduct getSelectedProduct() {return selectedProduct.get();}
402.
403.      /** setSelectedProduct sets the passed-in {@link PurchaseOrderProduct} newProduct
    value to the selectedProduct field.
404.       * @param newProduct The new {@link PurchaseOrderProduct} to store in selectedProduct.
405.       */
406.      public void setSelectedProduct(PurchaseOrderProduct newProduct)
    {selectedProduct.set(newProduct);}
407.
408.      /** selectedProductProperty returns the {@link ObjectProperty} of {@link
    PurchaseOrderProduct} selectedProduct
409.       *  field variable.
410.       * @return The {@link ObjectProperty} selectedProduct field.
411.       */
412.      public ObjectProperty<PurchaseOrderProduct> selectedProductProperty() {return
    selectedProduct;}
413.
414.      /** clearProducts clears the {@link ObservableList} products field.
415.       */
416.      public void clearProducts() {products.clear();}
417.
418.      /** productsProperty returns the {@link ObservableList} of {@link
    PurchaseOrderProduct} products field.
419.       * @return The {@link ObservableList} products field.
420.       */
421.      public ObservableList<PurchaseOrderProduct> productsProperty() {return products;}
422.
423.      /** getReturnQuantity returns the {@link Integer} value stored in the returnQuantity
    field.
424.       * @return The {@link Integer} value stored in returnQuantity.
425.       */
426.      public Integer getReturnQuantity() {return returnQuantity.getValue();}
427.
428.      /** setReturnQuantity sets the {@link Integer} value stored in the returnQuantity
    field.
429.       * @param newReturnQuantity The new {@link Integer} value to store in returnQuantity.
```

```
430.       */
431.      public void setReturnQuantity(Integer newReturnQuantity)
    {returnQuantity.setValue(newReturnQuantity);}
432.
433.      /** returnQuantityProperty returns the {@link IntegerProperty} returnQuantity field.
434.       * @return the {@link IntegerProperty} returnQuantity field.
435.       */
436.      public IntegerProperty returnQuantityProperty() {return returnQuantity;}
437.
438.      /** getReturnLabelTracker returns the {@link String} stored in returnLabelTracker.
439.       * @return The {@link String} stored in the returnLabelTracker field.
440.       */
441.      public String getReturnLabelTracker() {return returnLabelTracker.getValueSafe();}
442.
443.      /** setReturnLabelTracker sets the {@link String} value in the returnLabelTracker
    field to the passed-in
444.       *  {@link String} newReturnlabelTracker value.
445.       * @param newReturnLabelTracker The new {@link String} value to store in
    returnLabelTracker.
446.       */
447.      public void setReturnLabelTracker(String newReturnLabelTracker)
    {returnLabelTracker.setValue(newReturnLabelTracker);}
448.
449.      /** returnLabelTrackerProperty returns the {@link StringProperty} returnLabelTracker
    field.
450.       * @return The {@link StringProperty} returnLabelTracker field.
451.       */
452.      public StringProperty returnLabelTrackerProperty() {return returnLabelTracker;}
453.
454.      /** getInitialEvaluation returns the {@link String} value stored in the
    initialEvaluation field.
455.       * @return The {@link String} initialEvaluation field value.
456.       */
457.      public String getInitialEvaluation() {return initialEvaluation.getValueSafe();}
458.
459.      /** setInitialEvaluation sets the {@link String} value stored in the initialEvaluation
    field variable.
460.       * @param newInitialEvaluation The new {@link String} value to store in
    initialEvaluation.
461.       */
462.      public void setInitialEvaluation(String newInitialEvaluation)
    {initialEvaluation.setValue(newInitialEvaluation);}
463.
464.      /** initialEvaluationProperty returns the {@link StringProperty} initialEvaluation
    field.
465.       * @return The {@link StringProperty} initialEvaluation field variable.
466.       */
467.      public StringProperty initialEvaluationProperty() {return initialEvaluation;}
468.
469.      /** getSelectedDisposition returns the {@link String} value stored in the
    selectedDisposition field.
470.       * @return The {@link String} value stored in selectedDisposition.
471.       */
472.      public String getSelectedDisposition() {return selectedDisposition.getValueSafe();}
473.
474.      /** setSelectedDisposition sets the {@link String} value stored in the
    selectedDisposition field to the user-selected,
475.       *  passed-in {@link String} newSelectedDisposition parameter.
476.       * @param newSelectedDisposition The new {@link String} value to store in
    selectedDisposition.
477.       */
478.      public void setSelectedDisposition(String newSelectedDisposition)
    {selectedDisposition.setValue(newSelectedDisposition);}
479.
```

```
480.      /** selectedDispositionProperty returns the {@link StringProperty} selectedDisposition
    field variable.
481.       * @return The {@link StringProperty} selectedDisposition field.
482.       */
483.      public StringProperty selectedDispositionProperty() {return selectedDisposition;}
484.
485.      /** dispositionsProperty returns the {@link ObservableList} of {@link String}s of
    dispositions.
486.       * @return The {@link ObservableList} dispositions field variable.
487.       */
488.      public ObservableList<String> dispositionsProperty() {return dispositions;}
489.
490.      /** getDispositionNotes returns the {@link String} value stored in the
    dispositionNotes field.
491.       * @return The {@link String} value stored in dispositionNotes.
492.       */
493.      public String getDispositionNotes() {return dispositionNotes.getValueSafe();}
494.
495.      /** setDispositionNotes sets the {@link String} value stored in dispositionNotes to
    the user-entered {@link String}
496.       *  newDispositionNotes parameter.
497.       * @param newDispositionNotes The new {@link String} value to store in
    dispositionNotes.
498.       */
499.      public void setDispositionNotes(String newDispositionNotes)
    {dispositionNotes.setValue(newDispositionNotes);}
500.
501.      /** dispositionNotesProperty returns the {@link StringProperty} dispositionNotes
    field.
502.       * @return The {@link StringProperty} dispositionNotes field.
503.       */
504.      public StringProperty dispositionNotesProperty() {return dispositionNotes;}
505.
506.      /** getReplacementTrackingNumber returns the {@link String} tracking number for the
    replacement item(s) held in
507.       *  replacementTrackingNumber.
508.       * @return The {@link String} replacementTrackingNumber value.
509.       */
510.      public String getReplacementTrackingNumber() {return
    replacementTrackingNumber.getValueSafe();}
511.
512.      /** setReplacementTrackingNumber sets the {@link String} tracking number for the
    replacement product(s) into the
513.       *  replacementTrackingNumber field via the passed-in {@link String}
    newReplacementTrackingNumber.
514.       * @param newReplacementTrackingNumber The new {@link String} tracking number to store
    in replacementTrackingNumber.
515.       */
516.      public void setReplacementTrackingNumber(String newReplacementTrackingNumber)
    {replacementTrackingNumber.setValue(newReplacementTrackingNumber);}
517.
518.      /** replacementTrackingNumberProperty returns the {@link StringProperty} replacement
    tracking number field.
519.       * @return The {@link StringProperty} replacementTrackingNumber field.
520.       */
521.      public StringProperty replacementTrackingNumberProperty() {return
    replacementTrackingNumber;}
522.
523.      /** getReplacementShipDate returns the {@link LocalDate} stored in the
    replacementShipDate field.
524.       * @return The {@link LocalDate} replacementShipDate field value.
525.       */
526.      public LocalDate getReplacementShipDate() {return replacementShipDate.get();}
527.
```

```
528.      /** setReplacementShipDate sets the {@link LocalDate} value stored in the
    replacementShipDate field.
529.       * @param newReplacementShipDate The new {@link LocalDate} value to store for the
    replacementShipDate.
530.       */
531.      public void setReplacementShipDate(LocalDate newReplacementShipDate)
    {replacementShipDate.set(newReplacementShipDate);}
532.
533.      /** replacementShipDateProperty returns the {@link ObjectProperty} of {@link
    LocalDate} containing the
534.       *  replacement ship date.
535.       * @return The replacementShipDate {@link ObjectProperty} field.
536.       */
537.      public ObjectProperty<LocalDate> replacementShipDateProperty() {return
    replacementShipDate;}
538.
539.      /** getShipReplacementRepair returns the boolean value stored in the
    shipReplacementRepair field to indicate
540.       *  that we will be shipping a replacement or repair.
541.       * @return The boolean value stored in shipReplacementRepair.
542.       */
543.      public boolean getShipReplacementRepair() {return shipReplacementRepair.getValue();}
544.
545.      /** setShipReplacementRepair sets the indicator to ship the replacement or repair
    stored in shipReplacementRepair
546.       *  using the passed-in boolean value.
547.       * @param newShipReplacementRepair The boolean value to store in
    shipReplacementRepair.
548.       */
549.      public void setShipReplacementRepair(boolean newShipReplacementRepair)
    {shipReplacementRepair.setValue(newShipReplacementRepair);}
550.
551.      /** shipReplacementRepairProperty returns the {@link BooleanProperty}
    shipReplacementRepair field.
552.       * @return The {@link BooleanProperty} shipReplacementRepair field variable.
553.       */
554.      public BooleanProperty shipReplacementRepairProperty() {return shipReplacementRepair;}
555.
556.      /** convertHashMapToArrayList converts the passed-in {@link HashMap} into an {@link
    ArrayList}.
557.       * @param map The {@link HashMap} to convert into an {@link ArrayList}.
558.       * @return An {@link ArrayList} containing the list of key-value pairs stored in the
559.       * {@link HashMap} in "key value" format.
560.       */
561.      public ArrayList<String> convertHashMapToArrayList(HashMap<String, String> map){
562.          ArrayList<String> converted = new ArrayList<>();
563.          for(String key : map.keySet())
564.              converted.add(key + " " + map.get(key));
565.          return converted;
566.      }
567.  }
568.
```

RMADetailsFormController.java

```
1.  package RMA;
2.
3.  import javafx.application.Platform;
4.  import javafx.beans.InvalidationListener;
5.  import javafx.beans.Observable;
6.  import javafx.beans.binding.Bindings;
7.  import javafx.beans.value.ChangeListener;
8.  import javafx.beans.value.ObservableValue;
9.  import javafx.event.ActionEvent;
10. import javafx.event.Event;
11. import javafx.event.EventHandler;
12. import javafx.fxml.FXML;
13. import javafx.fxml.Initializable;
14. import javafx.scene.control.*;
15. import javafx.scene.layout.Region;
16. import javafx.util.converter.IntegerStringConverter;
17. import javafx.util.converter.NumberStringConverter;
18.
19. import java.net.URL;
20. import java.sql.SQLException;
21. import java.time.LocalDate;
22. import java.time.LocalDateTime;
23. import java.time.format.DateTimeFormatter;
24. import java.util.Optional;
25. import java.util.ResourceBundle;
26.
27.
28. /** RMADetailsFormController contains the logic and field references to the GUI controls on
    the RMADetailScreen.
29.  */
30. public class RMADetailsFormController implements Initializable {
31.     /** btnHome is used to return to the RMA List View screen.
32.      *  (Requirement 5.7)
33.      */
34.     @FXML
35.     private Button btnHome;
36.     /** pgRMAProgress is used to display how far along the RMA is to completion.
37.      * (Requirement 5.1)
38.      */
39.     @FXML
40.     private ProgressBar pgRMAProgress;
41.     /** tpInformation is used to expand collapse the Information section of the RMA Details
    form.
42.      *  (Requirement 5.2)
43.      */
44.     @FXML
45.     private TitledPane tpInformation;
46.     /** lblAutoGenNumber1 displays the {@link String} RMA ID.
47.      * (Requirement 5.2.2)
48.      */
49.     @FXML
50.     private Label lblAutoGenNumber1;
51.     /** hplCustomerNameDetails1 displays customer-specific information when clicked.
52.      */
53.     @FXML
54.     private Hyperlink hplCustomerNameDetails1;
55.     /** lblBusinessNameDetails1 holds the {@link String} Business Name referenced in this
    RMA request.
56.      */
57.     @FXML
58.     private Label lblBusinessNameDetails1;
```

```
59.     /** lblRmaAgeValue1 holds the number of days this RMA has been open since creation.
60.      * (Requirement 5.2.4)
61.      */
62.     @FXML
63.     private Label lblRMAAgeValue1;
64.     /** hplPoNumber1 displays details about the Purchase Order referenced in this RMA
    request.
65.      */
66.     @FXML
67.     private Hyperlink hplPONumber1;
68.     /** txtShippingAddress1 contains the Business's address details.
69.      */
70.     @FXML
71.     private TextArea txtShippingAddress1;
72.     /** lblAutoGenLastModifiedBy1 contains the date, time, and user that last modified this
    RMA request before opening.
73.      *(Requirement 5.2.3)
74.      */
75.     @FXML
76.     private Label lblAutoGenLastModifiedBy1;
77.     /** cmbEmployeeInfo1 contains the list of {@link String} usernames of employees stored
    in the database.
78.      */
79.     @FXML
80.     private ComboBox<String> cmbEmployeeInfo1;
81.     /** cmbRMAStatus1 contains the {@link String} list of RMA Statuses in the database.
82.      */
83.     @FXML
84.     private ComboBox<String> cmbRMAStatus1;
85.     /** cmbCreditReplaceRepair1 contains the {@link String} list of Credit, Replace, or
    Repair options.
86.      */
87.     @FXML
88.     private ComboBox<String> cmbCreditReplaceRepair1;
89.     /** cmbReasonCode1 contains the list of {@link String} Return Reason Codes in the
    database.
90.      */
91.     @FXML
92.     private ComboBox<String> cmbReasonCode1;
93.     /** txtSpecialInstruction1 contains additional info entered by an Admin or Analyst about
    the RMA request.
94.      */
95.     @FXML
96.     private TextArea txtSpecialInstruction1;
97.     /** lblAutoGenCreatedBy1 contains the username that created the RMA request.
98.      *  (Requirement 5.2.5)
99.      */
100.      @FXML
101.      private Label lblAutoGenCreatedBy1;
102.      /** btnSpecialInstructionEdit1 allows the user to edit and save changes to the
    txtSpecialInstruction1 {@link TextArea}.
103.       */
104.      @FXML
105.      private Button btnSpecialInstructionEdit1;
106.      /** tpProductInformation allows the user to expand and collapse the Product
    Information section of the RMA Details form.
107.       */
108.      @FXML
109.      private TitledPane tpProductInformation;
110.      /** cmbProductDetail1 contains the list of {@link PurchaseOrderProduct}s for the
    selected PO Number.
111.       */
112.      @FXML
113.      private ComboBox<PurchaseOrderProduct> cmbProductDetail1;
```

```
114.      /** txtReturnLabelTracking1 contains the return tracking label from the customer for
    the RMA.
115.       */
116.      @FXML
117.      private TextField txtReturnLabelTracking1;
118.      /** btnReturnLabelTrackingEdit1 allows the user to edit and save changes to the
    txtReturnLabelTracking1
119.       *  {@link TextField}.
120.       */
121.      @FXML
122.      private Button btnReturnLabelTrackingEdit1;
123.      /** txtQuantity1 holds the entered return quantity of the given Product for this RMA
    request.
124.       */
125.      @FXML
126.      private TextField txtQuantity1;
127.      /** btnQuantityEdit1 allows the user to edit and save changes to the txtQuantity1
    {@link TextField}.
128.       */
129.      @FXML
130.      private Button btnQuantityEdit1;
131.      /** tpProductEvaluation allows the user to expand and collapse the Product Evaluation
    section.
132.       * (Requirement 5.6)
133.       */
134.      @FXML
135.      private TitledPane tpProductEvaluation;
136.      /** txtInitialEvaluation1 contains the initial evaluation of the returned Product for
    this RMA.
137.       */
138.      @FXML
139.      private TextArea txtInitialEvaluation1;
140.      /** btnInitialEvaluationEdit1 allows the user to edit and save changes to the
    txtInitialEvaluation1 {@link TextField}.
141.       */
142.      @FXML
143.      private Button btnInitialEvaluationEdit1;
144.      /** txtEngineeringEvaluation1 contains the evaluation of the returned Product by a
    user in the Engineers group.
145.       */
146.      @FXML
147.      private TextArea txtEngineeringEvaluation1;
148.      /** btnEngineeringEvaluationEdit1 allows the user to edit and save changes to the
    txtEngineeringEvaluation1 {@link TextArea}.
149.       */
150.      @FXML
151.      private Button btnEngineeringEvaluationEdit1;
152.      /** tpProductDisposition allows the user to collapse and expand the Product
    Disposition section of the RMA Details form.
153.       */
154.      @FXML
155.      private TitledPane tpProductDisposition;
156.      /** cmbDisposition contains the {@link String} list of dispositions available in the
    database.
157.       */
158.      @FXML
159.      private ComboBox<String> cmbDisposition1;
160.      /** txtDispositionNotes1 contains any notes to further describe the selected
    disposition.
161.       */
162.      @FXML
163.      private TextArea txtDispositionNotes1;
164.      /** btnDispositionNotesEdit1 button allows the user to edit and save changes to the
    txtDispositionNotes1 {@link TextArea}.
165.       */
```

```
166.      @FXML
167.      private Button btnDispositionNotesEdit1;
168.      /** tpReplacementDetail allows the user to collapse and expand the Replacement Detail
    section of the RMA Details form.
169.       * (Requirement 5.3)
170.       */
171.      @FXML
172.      private TitledPane tpReplacementDetail;
173.      /** txtReplacementRepairTracking1 contains the shipping tracking number for the
    replacement or repair that will be
174.       *  sent back to the customer.
175.       */
176.      @FXML
177.      private TextField txtReplacementRepairTracking1;
178.      /** btnReplacementRepairTrackingEdit1 allows the user to edit and save changes to the
    txtReplacementRepairTracking1
179.       *  {@link TextArea}.
180.       */
181.      @FXML
182.      private Button btnReplacementRepairTrackingEdit1;
183.      /** dpReplacementRepairDate1 lets the user select the date the replacement or repair
    is being sent out for shipping.
184.       */
185.      @FXML
186.      private DatePicker dpReplacementRepairDate1;
187.      /** cbReplacementRepairShip1 is used to signal that the replacement or repair is being
    shipped.
188.       */
189.      @FXML
190.      private CheckBox cbReplacementRepairShip1;
191.      /** txtDetailsWindow contains any additional details about the selected Customer or PO
    Number.
192.       * (Requirement 5.4)
193.       */
194.      @FXML
195.      private TextArea txtDetailsWindow;
196.      /** model contains the {@link RMADetailsFormModel} backing this form.
197.       */
198.      private RMADetailsFormModel model;
199.      /** These doubles are used to resize the collapsed TitlePanes back to their initial
    height.
200.       */
201.      private double tpInformationPrefHeight;
202.      private double tpProductPrefHeight;
203.      private double tpEvaluationPrefHeight;
204.      private double tpDispositionPrefHeight;
205.      private double tpReplaceRepairPrefHeight;
206.      /** The following booleans are used to revert a change made to the corresponding
    {@link ComboBox} or other control.
207.       */
208.      private boolean ownerRevert = false;
209.      private boolean rmaStatusRevert = false;
210.      private boolean creditReplaceRepairRevert = false;
211.      private boolean returnReasonCodeRevert = false;
212.      private boolean productRevert = false;
213.      private boolean dispositionRevert = false;
214.      private boolean dpReplacementRepairDate1Revert = false;
215.      private boolean cbReplacementRepairShip1Revert = false;
216.      /** initialSetup is used to skip ChangeListeners when setting initial values while
    loading form.
217.       */
218.      private boolean initialSetup = true;
219.
220.
```

```
221.      /** Default empty constructor, used when {@link javafx.fxml.FXMLLoader} loads the fxml
    file.
222.       */
223.      public RMADetailsFormController(){}
224.
225.      /** loadRMADetails loads the form with the details for the passed-in RMA ID.
226.       *  (Requirement 5.2.4, 5.2.8, 5.4.1)
227.       * @param  rmaId The {@link String} RMA ID to load from the database.
228.       * @throws SQLException If there is an issue connecting to the SQL Server database or
    the SQL query.
229.       */
230.      public void loadRMADetails(String rmaId) throws SQLException {
231.          // Load model with the requested RMA's details.
232.          model.getRMADetails(rmaId);
233.
234.          // Set the age.
235.          model.updateAge();
236.
237.          // Set the current progress.
238.          model.updateRMAProgress();
239.
240.          // Populate txtDetailsWindow with the default initial output.
241.          hplCustomerNameDetails1.fire();
242.
243.          // Update all future changes.
244.          initialSetup = false;
245.      }
246.
247.      /** initialize ties together the GUI's elements with the RMAFormModel and
248.       *  (Requirement 5.2.6, 5.2.7, 5.2.7.1, 5.2.8.1, 5.2.8.1.1, 5.2.8.1.2, 5.2.8.2,
249.       *               5.3.1, 5.4.1, 5.5, 5.6.1, 5.6.2, 5.6.2.1, 5.6.3, 5.8)
250.       * @param location The {@link URL} location of the fxml file that describes
251.       *                 the RMADetailForm window elements and their layout.
252.       * @param resources {@link ResourceBundle} that contains locale-specific data.
253.       */
254.      @Override
255.      public void initialize(URL location, ResourceBundle resources) {
256.          // First create a new RMADetailsFormModel.
257.          model = new RMADetailsFormModel();
258.
259.          // Create TextFormatter for txtQuantity to restrict input.
260.          TextFormatter<String> textFormatter = new TextFormatter<>(change -> {
261.              if (!change.isContentChange())
262.                  return change;
263.
264.              String text = change.getControlNewText();
265.
266.              if (!text.matches("^\\d*"))
267.                  return null;
268.              else
269.                  return change;
270.          });
271.
272.          // Next bind the model to the GUI.
273.          pgRMAProgress.progressProperty().bindBidirectional(model.rmaProgressProperty());
274.          lblAutoGenNumber1.textProperty().bind(model.rmaIDProperty());
275.          hplCustomerNameDetails1.textProperty().bind(model.selectedCustomerNameProperty());
276.
    lblBusinessNameDetails1.textProperty().bind(model.selectedBusinessNameProperty().asString())
    ;
277.          lblRMAAgeValue1.textProperty().bind(model.ageProperty().asString());
278.          hplPONumber1.textProperty().bind(model.selectedPONumberProperty());
279.
    txtShippingAddress1.textProperty().bind(model.shippingAddressProperty().asString());
```

```
280.
    lblAutoGenLastModifiedBy1.textProperty().bind(Bindings.concat(model.lastModifiedByProperty()
    , ", ", model.lastModifiedProperty()));
281.            cmbEmployeeInfo1.valueProperty().bindBidirectional(model.selectedOwnerProperty());
282.            cmbEmployeeInfo1.setItems(model.ownersProperty());
283.
    cmbRMAStatus1.valueProperty().bindBidirectional(model.selectedRMAStatusProperty());
284.            cmbRMAStatus1.setItems(model.rmaStatusesProperty());
285.
    cmbCreditReplaceRepair1.valueProperty().bindBidirectional(model.selectedCreditReplaceRepairP
    roperty());
286.            cmbCreditReplaceRepair1.setItems(model.creditReplaceRepairProperty());
287.
    cmbReasonCode1.valueProperty().bindBidirectional(model.selectedReturnReasonCodeProperty());
288.            cmbReasonCode1.setItems(model.returnReasonCodesProperty());
289.
    txtSpecialInstruction1.textProperty().bindBidirectional(model.additionalInfoProperty());
290.
    lblAutoGenCreatedBy1.textProperty().bind(Bindings.concat(model.createdByProperty(), ", ",
    model.createdProperty()));
291.
    cmbProductDetail1.valueProperty().bindBidirectional(model.selectedProductProperty());
292.            cmbProductDetail1.setItems(model.productsProperty());
293.
    txtReturnLabelTracking1.textProperty().bindBidirectional(model.returnLabelTrackerProperty())
    ;
294.            txtQuantity1.textProperty().bindBidirectional(model.returnQuantityProperty(), new
    NumberStringConverter());
295.            txtQuantity1.setTextFormatter(textFormatter);
296.
    txtInitialEvaluation1.textProperty().bindBidirectional(model.initialEvaluationProperty());
297.
    txtEngineeringEvaluation1.textProperty().bindBidirectional(model.engineeringEvaluationProper
    ty());
298.
    cmbDisposition1.valueProperty().bindBidirectional(model.selectedDispositionProperty());
299.            cmbDisposition1.setItems(model.dispositionsProperty());
300.
    txtDispositionNotes1.textProperty().bindBidirectional(model.dispositionNotesProperty());
301.
    txtReplacementRepairTracking1.textProperty().bindBidirectional(model.replacementTrackingNumb
    erProperty());
302.
    dpReplacementRepairDate1.valueProperty().bindBidirectional(model.replacementShipDateProperty
    ());
303.
    cbReplacementRepairShip1.selectedProperty().bindBidirectional(model.shipReplacementRepairPro
    perty());
304.
    txtDetailsWindow.textProperty().bindBidirectional(model.detailsWindowTextProperty());
305.
306.            // Set selection permissions for Analyst and Engineer.
307.            if (DBService.getInstance().getRole().equals("engineers")) { // Disable all
    controls except btnEngineeringEvaluationEdit1.
308.                cmbEmployeeInfo1.setEditable(false);
309.                cmbEmployeeInfo1.getEditor().setEditable(false);
310.                cmbRMAStatus1.setEditable(false);
311.                cmbCreditReplaceRepair1.setEditable(false);
312.                cmbReasonCode1.setEditable(false);
313.                btnSpecialInstructionEdit1.setDisable(true);
314.                cmbProductDetail1.setEditable(false);
315.                btnReturnLabelTrackingEdit1.setDisable(true);
316.                btnQuantityEdit1.setDisable(true);
317.                btnInitialEvaluationEdit1.setDisable(true);
318.                cmbDisposition1.setEditable(false);
```

```
319.                btnDispositionNotesEdit1.setDisable(true);
320.                btnReplacementRepairTrackingEdit1.setDisable(true);
321.                dpReplacementRepairDate1.setEditable(false);
322.                cbReplacementRepairShip1.setDisable(true);
323.            } else if (DBService.getInstance().getRole().equals("analysts"))
324.                btnEngineeringEvaluationEdit1.setDisable(true);
325.
326.            // Add ChangeListeners to each ComboBox so that their updates push to the
    database.
327.            cmbEmployeeInfo1.valueProperty().addListener(
328.                new ChangeListener<String>() {
329.                    /** changed listens for changes to the cmbEmployeeInfo1 {@link ComboBox}
    in order to update them in the
330.                     *  back-end database.
331.                     * @param observableValue The {@link ObservableValue} that changed.
332.                     * @param oldValue The {@link String} previous value.
333.                     * @param newValue The {@link String} new value to commit to the database.
334.                     */
335.                    @Override
336.                    public void changed(ObservableValue<? extends String> observableValue,
    String oldValue, String newValue) {
337.                        if (!initialSetup) // Ignore initially setting value.
338.                            if (!ownerRevert) {
339.                                try {
340.                                    DBService.getInstance().updateRMAOwner(model.getRmaId(),
    newValue);
341.                                    updateLastModifiedDetails();
342.                                    model.updateRMAProgress();
343.                                } catch (SQLException e) {
344.                                    Alert a = new Alert(Alert.AlertType.ERROR);
345.                                    a.setContentText(
346.                                        "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
347.                                        e.getLocalizedMessage() + System.lineSeparator() +
348.                                        "Please select the value again to try again."
349.                                    );
350.                                    a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
351.                                    a.showAndWait();
352.                                    ownerRevert = true;
353.                                    model.setSelectedOwner(oldValue);
354.                                }
355.                            } else // Let the reversion go through.
356.                                ownerRevert = false;
357.                    }
358.                }
359.            );
360.
361.            cmbRMAStatus1.valueProperty().addListener(
362.                new ChangeListener<String>() {
363.                    /** changed listens for changes to the cmbRMAStatus1 {@link ComboBox}, in
    particular the "Closed" option,
364.                     *  to override and revert the decision if the RMA is not ready for
    closing, or confirm with the user
365.                     *  before committing the close, because nothing can be modified after
    committing it.
366.                     * @param observableValue The {@link ObservableValue} that changed.
367.                     * @param oldValue The {@link String} previous value.
368.                     * @param newValue The {@link String} new value.
369.                     */
370.                    @Override
371.                    public void changed(ObservableValue<? extends String> observableValue,
    String oldValue, String newValue) {
372.                        if (!initialSetup) // Ignore initially setting value.
373.                            if (!rmaStatusRevert) {
374.                                if (newValue.equals("Closed")) {
```

```
375.                                 if (model.getRMAProgress() == 1.0) {
376.                                     // Ask the user if they want to continue with closing
    the request.
377.                                     ButtonType confirm = new ButtonType("Confirm",
    ButtonBar.ButtonData.OK_DONE); // Change the button context from OK to Confirm
378.                                     Alert a = new Alert(
379.                                         Alert.AlertType.CONFIRMATION,
380.                                         "Closing an RMA request will mean no additional
    changes may be made to it. " +
381.                                         "Do you want to continue?",
382.                                         confirm,
383.                                         ButtonType.CANCEL
384.                                     );
385.                                     a.setTitle("Closing RMA Confirmation");
386.                                     a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
387.                                     Optional<ButtonType> result = a.showAndWait();
388.
389.                                     // Update database and disable all controls.
390.                                     if (result.isPresent() && result.get() == confirm) {
391.                                         try {
392.
    DBService.getInstance().updateRMAStatus(model.getRmaId(), newValue);
393.                                             updateLastModifiedDetails();
394.                                             cmbEmployeeInfo1.setDisable(true);
395.                                             cmbRMAStatus1.setDisable(true);
396.                                             cmbCreditReplaceRepair1.setDisable(true);
397.                                             cmbReasonCode1.setDisable(true);
398.                                             btnSpecialInstructionEdit1.setDisable(true);
399.                                             cmbProductDetail1.setDisable(true);
400.                                             btnReturnLabelTrackingEdit1.setDisable(true);
401.                                             btnQuantityEdit1.setDisable(true);
402.                                             btnInitialEvaluationEdit1.setDisable(true);
403.
    btnEngineeringEvaluationEdit1.setDisable(true);
404.                                             cmbDisposition1.setDisable(true);
405.                                             btnDispositionNotesEdit1.setDisable(true);
406.
    btnReplacementRepairTrackingEdit1.setDisable(true);
407.                                             dpReplacementRepairDate1.setDisable(true);
408.                                             cbReplacementRepairShip1.setDisable(true);
409.                                         } catch (SQLException e) {
410.                                             Alert error = new
    Alert(Alert.AlertType.ERROR);
411.                                             error.setContentText(
412.                                                 "Error while connecting to SQL Server
    database!" + System.lineSeparator() +
413.                                                 e.getLocalizedMessage() +
    System.lineSeparator() +
414.                                                 "Please select the value again to try
    again."
415.                                             );
416.
    error.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
417.                                             error.showAndWait();
418.                                             rmaStatusRevert = true;
419.                                             model.setSelectedRMAStatus(oldValue);
420.                                         }
421.                                     } else { // Revert the change.
422.                                         rmaStatusRevert = true;
423.                                         model.setSelectedRMAStatus(oldValue);
424.                                     }
425.                                 } else { // Notify the user about being unable to close at
    this time and revert the change.
426.                                     Alert a = new Alert(
427.                                         Alert.AlertType.ERROR,
```

```
                                        "The RMA is not completely filled out. Please set any missing fields and try again.",
                                        ButtonType.OK
                                    );
                                    a.setTitle("Closing RMA Error");
                                    a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
                                    Optional<ButtonType> result = a.showAndWait();
                                    rmaStatusRevert = true;
                                    model.setSelectedRMAStatus(oldValue);
                                }
                            } else { // Change can go through, update database.
                                try {
                                    DBService.getInstance().updateRMAStatus(model.getRmaId(), newValue);
                                    updateLastModifiedDetails();
                                    model.updateRMAProgress();
                                } catch (SQLException e) {
                                    Alert error = new Alert(Alert.AlertType.ERROR);
                                    error.setContentText(
                                        "Error while connecting to SQL Server database!" + System.lineSeparator() +
                                        e.getLocalizedMessage() + System.lineSeparator() +
                                        "Please select the value again to try again."
                                    );
                                    error.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
                                    error.showAndWait();
                                    rmaStatusRevert = true;
                                    model.setSelectedRMAStatus(oldValue);
                                }
                            }
                        } else // Let the reversion go through.
                            rmaStatusRevert = false;
                    }
                }
            );

            cmbCreditReplaceRepair1.valueProperty().addListener(
                new ChangeListener<String>() {
                    /** changed listens for changes to the cmbCreditReplaceRepair1 {@link ComboBox} in case the user selects
                     *  or de-selects the "Credit" option in order to clear, disable, and hide, or re-enable, the Replacement
                     *  Detail section.
                     * @param observableValue The {@link ObservableValue} that changed.
                     * @param oldValue The {@link String} previous value.
                     * @param newValue The {@link String} new value.
                     */
                    @Override
                    public void changed(ObservableValue<? extends String> observableValue, String oldValue, String newValue) {
                        if (!initialSetup) {
                            if (!creditReplaceRepairRevert) {
                                if (newValue.equals("Credit")) {
                                    // Ask the user if they want to confirm clearing the Replacement Detail section because
                                    // it is unnecessary if the customer is receiving a Credit.
                                    if ((!model.getReplacementTrackingNumber().isEmpty() && !model.getReplacementTrackingNumber().isBlank()) ||
                                        model.getReplacementShipDate() != null ||
                                        model.getShipReplacementRepair()
                                    ) {
                                        // Ask the user if they want to continue with clearing and disabling the fields.
```

```
482.                                    ButtonType confirm = new ButtonType("Confirm",
    ButtonBar.ButtonData.OK_DONE); // Change the button context from OK to Confirm
483.                                    Alert a = new Alert(
484.                                            Alert.AlertType.CONFIRMATION,
485.                                            "Selecting \"Credit\" will clear all data
    entered under the " +
486.                                                    "\"Replacement Information\" section
    and disable those controls " +
487.                                                    "because the customer will not be
    receiving a replacement or repair when " +
488.                                                    "they are receiving credit. Do you
    want to continue?",
489.                                            confirm,
490.                                            ButtonType.CANCEL
491.                                    );
492.                                    a.setTitle("Clearing Replacement Information
    Verification");
493.                                    a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
494.                                    Optional<ButtonType> result = a.showAndWait();
495.
496.                                    // Clear and disable the controls under Replacement
    Information.
497.                                    if (result.isPresent() && result.get() == confirm) {
498.                                        try {
499.
    DBService.getInstance().updateRMACreditReplaceRepair(model.getRmaId(), newValue);
500.
501.                                            model.setReplacementTrackingNumber("");
502.
    DBService.getInstance().updateRMAReplacementTrackingNumber(model.getRmaId(),
    model.getReplacementTrackingNumber());
503.
    txtReplacementRepairTracking1.setDisable(true);
504.
505.                                            model.setReplacementShipDate(null);
506.
    DBService.getInstance().updateRMAReplacementShipDate(model.getRmaId(),
    model.getReplacementShipDate());
507.                                            dpReplacementRepairDate1.setDisable(true);
508.
509.                                            model.setShipReplacementRepair(false);
510.
    DBService.getInstance().updateRMAShipReplacementRepair(model.getRmaId(),
    model.getShipReplacementRepair());
511.                                            cbReplacementRepairShip1.setDisable(true);
512.
513.                                            tpReplacementDetail.setExpanded(false);
514.                                            tpReplacementDetail.setDisable(true);
515.
516.                                            updateLastModifiedDetails();
517.                                            model.updateRMAProgress();
518.                                        } catch (SQLException e) {
519.                                            Alert error = new
    Alert(Alert.AlertType.ERROR);
520.                                            error.setContentText(
521.                                                    "Error while connecting to SQL Server
    database!" + System.lineSeparator() +
522.                                                            e.getLocalizedMessage() +
    System.lineSeparator() +
523.                                                            "Please select the value again
    to try again."
524.                                            );
525.
    error.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
526.                                            error.showAndWait();
```

```
527.                                            creditReplaceRepairRevert = true;
528.
    model.setSelectedCreditReplaceRepair(oldValue);
529.                                        }
530.                                    } else {// Set value back to its previous value.
531.                                        creditReplaceRepairRevert = true;
532.                                        model.setSelectedCreditReplaceRepair(oldValue);
533.                                    }
534.                                } else { // Update the value and disable the controls.
535.                                    try {
536.
    DBService.getInstance().updateRMACreditReplaceRepair(model.getRmaId(), newValue);
537.
538.                                        txtReplacementRepairTracking1.setDisable(true);
539.                                        dpReplacementRepairDate1.setDisable(true);
540.                                        cbReplacementRepairShip1.setDisable(true);
541.                                        tpReplacementDetail.setExpanded(false);
542.                                        tpReplacementDetail.setDisable(true);
543.
544.                                        updateLastModifiedDetails();
545.                                        model.updateRMAProgress();
546.                                    } catch (SQLException e) {
547.                                        Alert error = new Alert(Alert.AlertType.ERROR);
548.                                        error.setContentText(
549.                                                "Error while connecting to SQL Server
    database!" + System.lineSeparator() +
550.                                                        e.getLocalizedMessage() +
    System.lineSeparator() +
551.                                                        "Please select the value again to
    try again."
552.                                        );
553.
    error.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
554.                                        error.showAndWait();
555.                                        creditReplaceRepairRevert = true;
556.                                        model.setSelectedCreditReplaceRepair(oldValue);
557.                                    }
558.                                }
559.                            } else { // Re-enable the controls.
560.                                try {
561.
    DBService.getInstance().updateRMACreditReplaceRepair(model.getRmaId(), newValue);
562.
563.                                    txtReplacementRepairTracking1.setDisable(false);
564.                                    dpReplacementRepairDate1.setDisable(false);
565.                                    cbReplacementRepairShip1.setDisable(false);
566.                                    tpReplacementDetail.setDisable(false);
567.                                    tpReplacementDetail.setExpanded(true);
568.
569.                                    updateLastModifiedDetails();
570.                                    model.updateRMAProgress();
571.                                } catch (SQLException e) {
572.                                    Alert error = new Alert(Alert.AlertType.ERROR);
573.                                    error.setContentText(
574.                                            "Error while connecting to SQL Server
    database!" + System.lineSeparator() +
575.                                                    e.getLocalizedMessage() +
    System.lineSeparator() +
576.                                                    "Please select the value again to try
    again."
577.                                    );
578.
    error.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
579.                                    error.showAndWait();
580.                                    creditReplaceRepairRevert = true;
```

```
581.                                     model.setSelectedCreditReplaceRepair(oldValue);
582.                                 }
583.                             }
584.                         } else // Let the reversion go through.
585.                             creditReplaceRepairRevert = false;
586.                     } else if (newValue.equals("Credit")) { // Disable the controls.
587.                         txtReplacementRepairTracking1.setDisable(true);
588.                         dpReplacementRepairDate1.setDisable(true);
589.                         cbReplacementRepairShip1.setDisable(true);
590.                         tpReplacementDetail.setExpanded(false);
591.                         tpReplacementDetail.setDisable(true);
592.                     }
593.                 }
594.             }
595.         );
596.
597.         cmbReasonCode1.valueProperty().addListener(
598.             new ChangeListener<String>() {
599.                 /** changed listens for changes to cmbReasonCode1 {@link ComboBox} in
    order to push the changes to the
600.                  *  database.
601.                  * @param observableValue The {@link ObservableValue} that changed.
602.                  * @param oldValue The {@link String} previous value.
603.                  * @param newValue The {@link String} new value.
604.                  */
605.                 @Override
606.                 public void changed(ObservableValue<? extends String> observableValue,
    String oldValue, String newValue) {
607.                     if (!initialSetup) // Ignore initially setting value.
608.                         if (!returnReasonCodeRevert) {
609.                             try {
610.
    DBService.getInstance().updateRMAReturnReasonCode(model.getRmaId(), newValue.substring(0,
    newValue.indexOf(" ")));
611.                                 updateLastModifiedDetails();
612.                                 model.updateRMAProgress();
613.                             } catch (SQLException e) {
614.                                 Alert a = new Alert(Alert.AlertType.ERROR);
615.                                 a.setContentText(
616.                                     "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
617.                                     e.getLocalizedMessage() + System.lineSeparator() +
618.                                     "Please select the value again to try again."
619.                                 );
620.                                 a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
621.                                 a.showAndWait();
622.                                 returnReasonCodeRevert = true;
623.                                 model.setSelectedReturnReasonCode(oldValue);
624.                             }
625.                         } else // Let the reversion go through.
626.                             returnReasonCodeRevert = false;
627.                 }
628.             }
629.         );
630.
631.         cmbProductDetail1.valueProperty().addListener(
632.             new ChangeListener<PurchaseOrderProduct>() {
633.                 /** changed listens for changes to the cmbProductDetail1 {@link ComboBox}
    in order to push them to the
634.                  *  back-end database.
635.                  * @param observableValue The {@link ObservableValue} that changed.
636.                  * @param oldValue The {@link PurchaseOrderProduct} previous value.
637.                  * @param newValue The {@link PurchaseOrderProduct} new value.
638.                  */
639.                 @Override
```

```
640.                 public void changed(ObservableValue<? extends PurchaseOrderProduct>
    observableValue, PurchaseOrderProduct oldValue, PurchaseOrderProduct newValue) {
641.                     if (!initialSetup) // Ignore initially setting value.
642.                         if (!productRevert) {
643.                             try {
644.                                 DBService.getInstance().updateRMAProduct(model.getRmaId(),
    newValue.getPurchaseOrderProductId());
645.                                 updateLastModifiedDetails();
646.                                 model.updateRMAProgress();
647.                             } catch (SQLException e) {
648.                                 Alert a = new Alert(Alert.AlertType.ERROR);
649.                                 a.setContentText(
650.                                     "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
651.                                     e.getLocalizedMessage() + System.lineSeparator() +
652.                                     "Please select the value again to try again."
653.                                 );
654.                                 a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
655.                                 a.showAndWait();
656.                                 productRevert = true;
657.                                 model.setSelectedProduct(oldValue);
658.                             }
659.                         } else // Let the reversion go through.
660.                             productRevert = false;
661.                 }
662.             }
663.         );
664.
665.         cmbDisposition1.valueProperty().addListener(
666.             new ChangeListener<String>() {
667.                 /** changed listens for changes to the cmbDisposition1 {@link ComboBox} in
    order to push them to the
668.                  *  back-end database.
669.                  * @param observableValue The {@link ObservableValue} that changed.
670.                  * @param oldValue The {@link String} previous value.
671.                  * @param newValue The {@link String} new value.
672.                  */
673.                 @Override
674.                 public void changed(ObservableValue<? extends String> observableValue,
    String oldValue, String newValue) {
675.                     if (!initialSetup) // Ignore initially setting value.
676.                         if (!dispositionRevert) {
677.                             try {
678.
    DBService.getInstance().updateRMADisposition(model.getRmaId(), newValue);
679.                                 updateLastModifiedDetails();
680.                                 model.updateRMAProgress();
681.                             } catch (SQLException e) {
682.                                 Alert a = new Alert(Alert.AlertType.ERROR);
683.                                 a.setContentText(
684.                                     "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
685.                                     e.getLocalizedMessage() + System.lineSeparator() +
686.                                     "Please select the value again to try again."
687.                                 );
688.                                 a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
689.                                 a.showAndWait();
690.                                 dispositionRevert = true;
691.                                 model.setSelectedDisposition(oldValue);
692.                             }
693.                         } else // Let the reversion go through.
694.                             dispositionRevert = false;
695.                 }
696.             }
697.         );
```

```
698.
699.          pgRMAProgress.progressProperty().addListener(
700.              new ChangeListener<Number>() {
701.                  /** changed listens for changes to the pgRMAProgress {@link ProgressBar}
    value in order to notify the
702.                   *  user that the RMA request can be closed.
703.                   * @param observableValue The {@link ObservableValue} that was changed.
704.                   * @param oldValue The {@link Number} previous value.
705.                   * @param newValue The {@link Number} new value.
706.                   */
707.                  @Override
708.                  public void changed(ObservableValue<? extends Number> observableValue,
    Number oldValue, Number newValue) {
709.                      // Notify the user that they can close the RMA request.
710.                      if (newValue.doubleValue() == 1.0 &&
    !model.getSelectedRMAStatus().equals("Closed")) {
711.                          Alert a = new Alert(
712.                              Alert.AlertType.CONFIRMATION,
713.                              "The RMA Request is ready to be closed (select \"Closed\"
    under RMA Status).",
714.                              ButtonType.OK
715.                          );
716.                          a.setTitle("RMA Ready to be Closed");
717.                          a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
718.                          a.showAndWait();
719.                      }
720.                  }
721.              }
722.          );
723.
724.          // Add ChangeListener to cbReplaceRepairShip1 to update RMA Progress and Last
    Modified details.
725.          cbReplacementRepairShip1.selectedProperty().addListener(
726.              new ChangeListener<Boolean>() {
727.                  /** changed listens for changes to cbReplacementRepairShip1 so that the
    Last Modified and RMA Progress
728.                   *  details are updated and any changes are pushed to the database.
729.                   * @param observableValue The {@link ObservableValue} that was changed.
730.                   * @param oldValue The {@link Boolean} previous value.
731.                   * @param newValue The {@link Boolean} new value.
732.                   */
733.                  @Override
734.                  public void changed(ObservableValue<? extends Boolean> observableValue,
    Boolean oldValue, Boolean newValue) {
735.                      if (!initialSetup) // Ignore initial values set during detail
    population.
736.                          if (!cbReplacementRepairShip1Revert)
737.                              try {
738.
    DBService.getInstance().updateRMAShipReplacementRepair(model.getRmaId(), newValue);
739.                                  updateLastModifiedDetails();
740.                                  model.updateRMAProgress();
741.                              } catch (SQLException e) {
742.                                  Alert a = new Alert(Alert.AlertType.ERROR);
743.                                  a.setContentText(
744.                                      "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
745.                                      e.getLocalizedMessage() + System.lineSeparator() +
746.                                      "Please select the value again to try again."
747.                                  );
748.                                  a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
749.                                  a.showAndWait();
750.                                  cbReplacementRepairShip1Revert = true;
751.                                  model.setShipReplacementRepair(oldValue);
752.                              }
```

```
753.                         else // Let the change go through.
754.                             cbReplacementRepairShip1Revert = false;
755.                     }
756.                 }
757.             );
758.
759.             // Add ChangeListener to dpReplacementRepairDate1 to update RMA Progress and Last
    Modified details.
760.             dpReplacementRepairDate1.valueProperty().addListener(
761.                 new ChangeListener<LocalDate>() {
762.                     /** changed listens for changes to dpReplacementRepairDate1 so that the
    Last Modified and RMA Progress
763.                      *  details are updated and any changes are pushed to the database.
764.                      * @param observableValue The {@link ObservableValue} that was changed.
765.                      * @param oldValue The {@link LocalDate} previous value.
766.                      * @param newValue The {@link LocalDate} new value.
767.                      */
768.                     @Override
769.                     public void changed(ObservableValue<? extends LocalDate> observableValue,
    LocalDate oldValue, LocalDate newValue) {
770.                         if (!initialSetup) // Ignore initial setup setting values.
771.                             if (!dpReplacementRepairDate1Revert)
772.                                 try {
773.
    DBService.getInstance().updateRMAReplacementShipDate(model.getRmaId(), newValue);
774.                                     updateLastModifiedDetails();
775.                                     model.updateRMAProgress();
776.                                 } catch (SQLException e) {
777.                                     Alert a = new Alert(Alert.AlertType.ERROR);
778.                                     a.setContentText(
779.                                         "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
780.                                         e.getLocalizedMessage() + System.lineSeparator() +
781.                                         "Please select the value again to try again."
782.                                     );
783.                                     a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
784.                                     a.showAndWait();
785.                                     dpReplacementRepairDate1Revert = true;
786.                                     model.setReplacementShipDate(oldValue);
787.                                 }
788.                             else // Let the reversion go through.
789.                                 dpReplacementRepairDate1Revert = false;
790.                     }
791.                 }
792.             );
793.             // Add another listener to dpReplacementRepairDate1 for when the user clears the
    TextField portion.
794.             dpReplacementRepairDate1.getEditor().textProperty().addListener(
795.                 new ChangeListener<String>() {
796.                     /** changed listens for changes to the {@link TextField} portion of the
    {@link DatePicker} for when
797.                      *  the user clears the field so that the actual stored value for the
    value is cleared.
798.                      * @param observable The {@link ObservableValue} that changed.
799.                      * @param oldValue The {@link String} previous value.
800.                      * @param newValue The {@link String} new value.
801.                      */
802.                     @Override
803.                     public void changed(final ObservableValue<? extends String> observable,
    String oldValue, String newValue) {
804.                         if (!initialSetup)
805.                             if (newValue.isEmpty())
806.                                 dpReplacementRepairDate1.setValue(null);
807.                     }
808.                 }
```

```
809.          );
810.
811.          // ChangeListeners that set the PrefHeight for each TitledPane to 0 if not
    expanded.
812.          tpInformation.expandedProperty().addListener(
813.              new ChangeListener<Boolean>() {
814.                  /** changed listens for the {@link TitledPane}s being collapsed and
    expanded in order to visually make
815.                   *  the collapsed pane look fully closed, and returning the height to the
    original height when expanded.
816.                   * @param observableValue The {@link ObservableValue} that changed.
817.                   * @param oldValue The {@link Boolean} previous value.
818.                   * @param newValue The {@link Boolean} new value.
819.                   */
820.                  @Override
821.                  public void changed(ObservableValue<? extends Boolean> observableValue,
    Boolean oldValue, Boolean newValue) {
822.                      tpInformation.setMaxHeight(!newValue ? 0.0 : tpInformationPrefHeight);
823.                  }
824.              }
825.          );
826.          tpProductInformation.expandedProperty().addListener(
827.              new ChangeListener<Boolean>() {
828.                  /** changed listens for the {@link TitledPane}s being collapsed and
    expanded in order to visually make
829.                   *  the collapsed pane look fully closed, and returning the height to the
    original height when expanded.
830.                   * @param observableValue The {@link ObservableValue} that changed.
831.                   * @param oldValue The {@link Boolean} previous value.
832.                   * @param newValue The {@link Boolean} new value.
833.                   */
834.                  @Override
835.                  public void changed(ObservableValue<? extends Boolean> observableValue,
    Boolean oldValue, Boolean newValue) {
836.                      tpProductInformation.setMaxHeight(!newValue ? 0.0 :
    tpProductPrefHeight);
837.                  }
838.              }
839.          );
840.          tpProductEvaluation.expandedProperty().addListener(
841.              new ChangeListener<Boolean>() {
842.                  /** changed listens for the {@link TitledPane}s being collapsed and
    expanded in order to visually make
843.                   *  the collapsed pane look fully closed, and returning the height to the
    original height when expanded.
844.                   * @param observableValue The {@link ObservableValue} that changed.
845.                   * @param oldValue The {@link Boolean} previous value.
846.                   * @param newValue The {@link Boolean} new value.
847.                   */
848.                  @Override
849.                  public void changed(ObservableValue<? extends Boolean> observableValue,
    Boolean oldValue, Boolean newValue) {
850.                      tpProductEvaluation.setMaxHeight(!newValue ? 0.0 :
    tpEvaluationPrefHeight);
851.                  }
852.              }
853.          );
854.          tpProductDisposition.expandedProperty().addListener(
855.              new ChangeListener<Boolean>() {
856.                  /** changed listens for the {@link TitledPane}s being collapsed and
    expanded in order to visually make
857.                   *  the collapsed pane look fully closed, and returning the height to the
    original height when expanded.
858.                   * @param observableValue The {@link ObservableValue} that changed.
859.                   * @param oldValue The {@link Boolean} previous value.
```

```
860.                     * @param newValue The {@link Boolean} new value.
861.                     */
862.                    @Override
863.                    public void changed(ObservableValue<? extends Boolean> observableValue,
    Boolean oldValue, Boolean newValue) {
864.                        tpProductDisposition.setMaxHeight(!newValue ? 0.0 :
    tpDispositionPrefHeight);
865.                    }
866.                }
867.            );
868.            tpReplacementDetail.expandedProperty().addListener(
869.                new ChangeListener<Boolean>() {
870.                    /** changed listens for the {@link TitledPane}s being collapsed and
    expanded in order to visually make
871.                     *  the collapsed pane look fully closed, and returning the height to the
    original height when expanded.
872.                     * @param observableValue The {@link ObservableValue} that changed.
873.                     * @param oldValue The {@link Boolean} previous value.
874.                     * @param newValue The {@link Boolean} new value.
875.                     */
876.                    @Override
877.                    public void changed(ObservableValue<? extends Boolean> observableValue,
    Boolean oldValue, Boolean newValue) {
878.                        tpReplacementDetail.setMaxHeight(!newValue ? 0.0 :
    tpReplaceRepairPrefHeight);
879.                    }
880.                }
881.            );
882.
883.            // Set the height to reset the TitledPanes' height on expansion.
884.            tpInformationPrefHeight = tpInformation.getPrefHeight();
885.            tpProductPrefHeight = tpProductInformation.getPrefHeight();
886.            tpEvaluationPrefHeight = tpProductEvaluation.getPrefHeight();
887.            tpDispositionPrefHeight = tpProductDisposition.getPrefHeight();
888.            tpReplaceRepairPrefHeight = tpReplacementDetail.getPrefHeight();
889.
890.            // Set each ComboBox's OnShowing method and the DatePicker's OnShowing method to
    close if the user is an engineer.
891.            cmbEmployeeInfo1.setOnShowing(
892.                new EventHandler<Event>() {
893.                    /** handle checks whether the user is an Engineer, and if so, prevents
    them from opening the {@link ComboBox}.
894.                     * @param event The {@link Event} passed to this event handler.
895.                     */
896.                    @Override
897.                    public void handle(Event event) {
898.                        if (DBService.getInstance().getRole().equals("engineers"))
899.                            Platform.runLater(() -> cmbEmployeeInfo1.hide());
900.                    }
901.                }
902.            );
903.            cmbRMAStatus1.setOnShowing(
904.                new EventHandler<Event>() {
905.                    /** handle checks whether the user is an Engineer, and if so, prevents
    them from opening the {@link ComboBox}.
906.                     * @param event The {@link Event} passed to this event handler.
907.                     */
908.                    @Override
909.                    public void handle(Event event) {
910.                        if (DBService.getInstance().getRole().equals("engineers"))
911.                            Platform.runLater(() -> cmbRMAStatus1.hide());
912.                    }
913.                }
914.            );
915.            cmbCreditReplaceRepair1.setOnShowing(
```

```
916.                new EventHandler<Event>() {
917.                    /** handle checks whether the user is an Engineer, and if so, prevents
    them from opening the {@link ComboBox}.
918.                     * @param event The {@link Event} passed to this event handler.
919.                     */
920.                    @Override
921.                    public void handle(Event event) {
922.                        if (DBService.getInstance().getRole().equals("engineers"))
923.                            Platform.runLater(() -> cmbCreditReplaceRepair1.hide());
924.                    }
925.                }
926.            );
927.            cmbReasonCode1.setOnShowing(
928.                new EventHandler<Event>() {
929.                    /** handle checks whether the user is an Engineer, and if so, prevents
    them from opening the {@link ComboBox}.
930.                     * @param event The {@link Event} passed to this event handler.
931.                     */
932.                    @Override
933.                    public void handle(Event event) {
934.                        if (DBService.getInstance().getRole().equals("engineers"))
935.                            Platform.runLater(() -> cmbReasonCode1.hide());
936.                    }
937.                }
938.            );
939.            cmbProductDetail1.setOnShowing(
940.                new EventHandler<Event>() {
941.                    /** handle checks whether the user is an Engineer, and if so, prevents
    them from opening the {@link ComboBox}.
942.                     * @param event The {@link Event} passed to this event handler.
943.                     */
944.                    @Override
945.                    public void handle(Event event) {
946.                        if (DBService.getInstance().getRole().equals("engineers"))
947.                            Platform.runLater(() -> cmbProductDetail1.hide());
948.                    }
949.                }
950.            );
951.            cmbDisposition1.setOnShowing(
952.                new EventHandler<Event>() {
953.                    /** handle checks whether the user is an Engineer, and if so, prevents
    them from opening the {@link ComboBox}.
954.                     * @param event The {@link Event} passed to this event handler.
955.                     */
956.                    @Override
957.                    public void handle(Event event) {
958.                        if (DBService.getInstance().getRole().equals("engineers"))
959.                            Platform.runLater(() -> cmbDisposition1.hide());
960.                    }
961.                }
962.            );
963.            dpReplacementRepairDate1.setOnShowing(
964.                new EventHandler<Event>() {
965.                    /** handle checks whether the user is an Engineer, and if so, prevents
    them from opening the {@link ComboBox}.
966.                     * @param event The {@link Event} passed to this event handler.
967.                     */
968.                    @Override
969.                    public void handle(Event event) {
970.                        if (DBService.getInstance().getRole().equals("engineers"))
971.                            Platform.runLater(() -> dpReplacementRepairDate1.hide());
972.                    }
973.                }
974.            );
975.        }
```

```
976.
977.      /** btnHomeOnAction is used to close the RMA request and return the user to the RMA
    List View.
978.       * @param event The {@link ActionEvent} passed to this event handler.
979.       */
980.      @FXML
981.      protected void btnHomeOnAction(ActionEvent event) {
982.          // Ask the user if they want to close.
983.          ButtonType confirm = new ButtonType("Confirm", ButtonBar.ButtonData.OK_DONE); //
    Change the button text from OK to Confirm
984.          Alert a = new Alert(
985.              Alert.AlertType.CONFIRMATION,
986.              "Are you sure you want to exit?",
987.              confirm,
988.              ButtonType.NO
989.          );
990.          a.setTitle("RMA Details Close Confirmation");
991.          a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
992.          Optional<ButtonType> result = a.showAndWait();
993.
994.          if (result.isPresent() && result.get() == confirm)
995.              btnHome.getScene().getWindow().hide();
996.      }
997.
998.      /** hplCustomerNameDetails1OnAction fires when the hplCustomerNameDetails1 is
    selected.
999.       * @param event The {@link ActionEvent} passed to this event handler.
1000.      * (Requirement 5.4.2, 5.5)
1001.      */
1002.     @FXML
1003.     protected void hplCustomerNameDetails1OnAction(ActionEvent event) {
1004.         String result = model.getShippingAddress() + System.lineSeparator() +
1005.                 "PO Numbers:" + System.lineSeparator();
1006.
1007.         for (String po : model.poNumbersProperty())
1008.             result += po + System.lineSeparator();
1009.
1010.         model.setDetailsWindowText(result);
1011.     }
1012.
1013.     /** hplPONumber1OnAction fires when the hplPONumber1 is selected.
1014.      * @param event The {@link ActionEvent} passed to this event handler.
1015.      * (Requirement 5.4.2, 5.5)
1016.      */
1017.     @FXML
1018.     protected void hplPONumber1OnAction(ActionEvent event) {
1019.         String result = "";
1020.         for (PurchaseOrderProduct product : model.productsProperty())
1021.             result +=
1022.                 "Product PO Number: " + product.getPONumber() + System.lineSeparator() +
1023.                 "Product Category: " + product.getProductCategory() +
    System.lineSeparator() +
1024.                 "Product Name: " + product.getProductName() + System.lineSeparator() +
1025.                 "Quantity Ordered: " + product.getQuantity() + System.lineSeparator() +
1026.                 "Order Date: " + product.getOrderDate() + System.lineSeparator() +
1027.                 "Deliver Date: " + product.getDeliverDate() + System.lineSeparator()
1028.             ;
1029.
1030.         model.setDetailsWindowText(result);
1031.     }
1032.
1033.     /** btnSpecialInstructionEdit1OnAction first asks the user whether they want to edit
    the {@link TextArea}, and if so,
1034.      *  changes the txtSpecialInstruction1 {@link TextArea} to editable, the "Edit" button
    to "Save", and disables
```

```
1035.      *  all other modifiable (or used to modify) controls while the editing is in
    progress. Once the user clicks on
1036.      *  "Save", the new contents of the {@link TextArea} will be pushed to the database,
    the disabled controls will be
1037.      *  re-enabled, and the button text will revert back to "Edit", along with changing
    the {@link TextArea} back to
1038.      *  non-editable.
1039.      * @param event The {@link ActionEvent} that is passed to this event handler.
1040.      * (Requirement 5.2.1, 5.2.1.1, 5.2.1.2, 5.2.1.3)
1041.      */
1042.     @FXML
1043.     protected void btnSpecialInstructionEdit1OnAction(ActionEvent event) {
1044.         if (btnSpecialInstructionEdit1.getText().equals("Edit")) {
1045.             // Ask the user if they want to edit the field.
1046.             if (editConfirmationMessage(btnSpecialInstructionEdit1, "Additional
    Info/Special Instruction", "Save")) {
1047.                 // Set field to be editable and disable the other modifiable (or used to
    modify) controls.
1048.                 txtSpecialInstruction1.setEditable(true);
1049.                 cmbEmployeeInfo1.setDisable(true);
1050.                 cmbRMAStatus1.setDisable(true);
1051.                 cmbCreditReplaceRepair1.setDisable(true);
1052.                 cmbReasonCode1.setDisable(true);
1053.                 cmbProductDetail1.setDisable(true);
1054.                 btnReturnLabelTrackingEdit1.setDisable(true);
1055.                 btnQuantityEdit1.setDisable(true);
1056.                 btnInitialEvaluationEdit1.setDisable(true);
1057.                 btnEngineeringEvaluationEdit1.setDisable(true);
1058.                 cmbDisposition1.setDisable(true);
1059.                 btnDispositionNotesEdit1.setDisable(true);
1060.                 if (!model.getSelectedCreditReplaceRepair().equals("Credit")) {
1061.                     btnReplacementRepairTrackingEdit1.setDisable(true);
1062.                     dpReplacementRepairDate1.setDisable(true);
1063.                     cbReplacementRepairShip1.setDisable(true);
1064.                 }
1065.             }
1066.         } else {
1067.             // Ask the user if they want to save the field.
1068.             if (editConfirmationMessage(btnSpecialInstructionEdit1, "Additional
    Info/Special Instruction", "Edit")) {
1069.                 try {
1070.                     // Update the field value in the database.
1071.                     DBService.getInstance().updateRMAAdditionalInfo(model.getRmaId(),
    model.getAdditionalInfo());
1072.                     updateLastModifiedDetails();
1073.                     model.updateRMAProgress();
1074.
1075.                     // Set field to be non-editable and enable the other modifiable (or
    used to modify) controls.
1076.                     txtSpecialInstruction1.setEditable(false);
1077.                     cmbEmployeeInfo1.setDisable(false);
1078.                     cmbRMAStatus1.setDisable(false);
1079.                     cmbCreditReplaceRepair1.setDisable(false);
1080.                     cmbReasonCode1.setDisable(false);
1081.                     cmbProductDetail1.setDisable(false);
1082.                     btnReturnLabelTrackingEdit1.setDisable(false);
1083.                     btnQuantityEdit1.setDisable(false);
1084.                     btnInitialEvaluationEdit1.setDisable(false);
1085.                     if (DBService.getInstance().getRole().equals("admins"))
1086.                         btnEngineeringEvaluationEdit1.setDisable(false);
1087.                     cmbDisposition1.setDisable(false);
1088.                     btnDispositionNotesEdit1.setDisable(false);
1089.                     if (!model.getSelectedCreditReplaceRepair().equals("Credit")) {
1090.                         btnReplacementRepairTrackingEdit1.setDisable(false);
1091.                         dpReplacementRepairDate1.setDisable(false);
```

```
1092.                         cbReplacementRepairShip1.setDisable(false);
1093.                     }
1094.                 } catch (SQLException e) {
1095.                     Alert a = new Alert(Alert.AlertType.ERROR);
1096.                     a.setContentText(
1097.                         "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
1098.                         e.getLocalizedMessage() + System.lineSeparator() +
1099.                         "Please click \"Save\" to try again."
1100.                     );
1101.                     a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
1102.                     a.showAndWait();
1103.                 }
1104.             }
1105.         }
1106.     }
1107.
1108.     /** btnReturnLabelTrackingEdit1OnAction first asks the user whether they want to edit
    the {@link TextField}, and if
1109.      *  so, changes the txtReturnLabelTrackingEdit1 {@link TextField} to editable, the
    "Edit" button to "Save", and
1110.      *  disables all other modifiable (or used to modify) controls while the editing is in
    progress. Once the user
1111.      *  clicks on "Save", the new contents of the {@link TextField} will be pushed to the
    database, the disabled controls
1112.      *  will be re-enabled, and the button text will revert back to "Edit", along with
    changing the {@link TextField}
1113.      *  back to non-editable.
1114.      * (Requirement 5.2.1, 5.2.1.1, 5.2.1.2, 5.2.1.3)
1115.      * @param event The {@link ActionEvent} that was passed to this method.
1116.      */
1117.     @FXML
1118.     protected void btnReturnLabelTrackingEdit1OnAction(ActionEvent event) {
1119.         if (btnReturnLabelTrackingEdit1.getText().equals("Edit")) {
1120.             // Ask the user if they want to edit the field.
1121.             if (editConfirmationMessage(btnReturnLabelTrackingEdit1, "Return Label
    Tracking #", "Save")) {
1122.                 // Set field to be editable and disable the other modifiable (or used to
    modify) controls.
1123.                 txtReturnLabelTracking1.setEditable(true);
1124.                 cmbEmployeeInfo1.setDisable(true);
1125.                 cmbRMAStatus1.setDisable(true);
1126.                 cmbCreditReplaceRepair1.setDisable(true);
1127.                 cmbReasonCode1.setDisable(true);
1128.                 btnSpecialInstructionEdit1.setDisable(true);
1129.                 cmbProductDetail1.setDisable(true);
1130.                 btnQuantityEdit1.setDisable(true);
1131.                 btnInitialEvaluationEdit1.setDisable(true);
1132.                 btnEngineeringEvaluationEdit1.setDisable(true);
1133.                 cmbDisposition1.setDisable(true);
1134.                 btnDispositionNotesEdit1.setDisable(true);
1135.                 if (!model.getSelectedCreditReplaceRepair().equals("Credit")) {
1136.                     btnReplacementRepairTrackingEdit1.setDisable(true);
1137.                     dpReplacementRepairDate1.setDisable(true);
1138.                     cbReplacementRepairShip1.setDisable(true);
1139.                 }
1140.             }
1141.         } else {
1142.             // Ask the user if they want to save the field.
1143.             if (editConfirmationMessage(btnReturnLabelTrackingEdit1, "Return Label
    Tracking #", "Edit")) {
1144.                 try {
1145.                     // Update the field value in the database.
1146.                     DBService.getInstance().updateRMAReturnLabelTracker(model.getRmaId(),
    model.getReturnLabelTracker());
```

```
1147.                     updateLastModifiedDetails();
1148.                     model.updateRMAProgress();
1149.
1150.                     // Set field to be non-editable and enable the other modifiable (or
    used to modify) controls.
1151.                     txtReturnLabelTracking1.setEditable(false);
1152.                     cmbEmployeeInfo1.setDisable(false);
1153.                     cmbRMAStatus1.setDisable(false);
1154.                     cmbCreditReplaceRepair1.setDisable(false);
1155.                     cmbReasonCode1.setDisable(false);
1156.                     btnSpecialInstructionEdit1.setDisable(false);
1157.                     cmbProductDetail1.setDisable(false);
1158.                     btnQuantityEdit1.setDisable(false);
1159.                     btnInitialEvaluationEdit1.setDisable(false);
1160.                     if (DBService.getInstance().getRole().equals("admins"))
1161.                         btnEngineeringEvaluationEdit1.setDisable(false);
1162.                     cmbDisposition1.setDisable(false);
1163.                     btnDispositionNotesEdit1.setDisable(false);
1164.                     if (!model.getSelectedCreditReplaceRepair().equals("Credit")) {
1165.                         btnReplacementRepairTrackingEdit1.setDisable(false);
1166.                         dpReplacementRepairDate1.setDisable(false);
1167.                         cbReplacementRepairShip1.setDisable(false);
1168.                     }
1169.                 } catch (SQLException e) {
1170.                     Alert a = new Alert(Alert.AlertType.ERROR);
1171.                     a.setContentText(
1172.                         "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
1173.                         e.getLocalizedMessage() + System.lineSeparator() +
1174.                         "Please click \"Save\" to try again."
1175.                     );
1176.                     a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
1177.                     a.showAndWait();
1178.                 }
1179.             }
1180.         }
1181.     }
1182.
1183.     /** btnQuantityEdit1OnAction first asks the user whether they want to edit the {@link
    TextField}, and if
1184.      *  so, changes the txtQuantity1 {@link TextField} to editable, the "Edit" button to
    "Save", and disables all other
1185.      *  modifiable (or used to modify) controls while the editing is in progress. Once the
    user clicks on "Save", the
1186.      *  new contents of the {@link TextField} will be pushed to the database, the disabled
    controls will be re-enabled,
1187.      *  and the button text will revert back to "Edit", along with changing the {@link
    TextField} back to non-editable.
1188.      * (Requirement 5.2.1, 5.2.1.1, 5.2.1.2, 5.2.1.3)
1189.      * @param event The {@link ActionEvent} that was passed to this method.
1190.      */
1191.     @FXML
1192.     protected void btnQuantityEdit1OnAction(ActionEvent event) {
1193.         if (btnQuantityEdit1.getText().equals("Edit")) {
1194.             // Ask the user if they want to edit the field.
1195.             if (editConfirmationMessage(btnQuantityEdit1, "Quantity", "Save")) {
1196.                 // Set field to be editable and disable the other modifiable (or used to
    modify) controls.
1197.                 txtQuantity1.setEditable(true);
1198.                 cmbEmployeeInfo1.setDisable(true);
1199.                 cmbRMAStatus1.setDisable(true);
1200.                 cmbCreditReplaceRepair1.setDisable(true);
1201.                 cmbReasonCode1.setDisable(true);
1202.                 btnSpecialInstructionEdit1.setDisable(true);
1203.                 cmbProductDetail1.setDisable(true);
```

```
1204.                 btnReturnLabelTrackingEdit1.setDisable(true);
1205.                 btnInitialEvaluationEdit1.setDisable(true);
1206.                 btnEngineeringEvaluationEdit1.setDisable(true);
1207.                 cmbDisposition1.setDisable(true);
1208.                 btnDispositionNotesEdit1.setDisable(true);
1209.                 if (!model.getSelectedCreditReplaceRepair().equals("Credit")) {
1210.                     btnReplacementRepairTrackingEdit1.setDisable(true);
1211.                     dpReplacementRepairDate1.setDisable(true);
1212.                     cbReplacementRepairShip1.setDisable(true);
1213.                 }
1214.             }
1215.         } else {
1216.             // Ask the user if they want to save the field.
1217.             if (editConfirmationMessage(btnQuantityEdit1, "Quantity", "Edit")) {
1218.                 try {
1219.                     // Update the field value in the database.
1220.                     DBService.getInstance().updateRMAReturnQuantity(model.getRmaId(),
    model.getReturnQuantity());
1221.                     updateLastModifiedDetails();
1222.                     model.updateRMAProgress();
1223.
1224.                     // Set field to be non-editable and enable the other modifiable (or
    used to modify) controls.
1225.                     txtQuantity1.setEditable(false);
1226.                     cmbEmployeeInfo1.setDisable(false);
1227.                     cmbRMAStatus1.setDisable(false);
1228.                     cmbCreditReplaceRepair1.setDisable(false);
1229.                     cmbReasonCode1.setDisable(false);
1230.                     btnSpecialInstructionEdit1.setDisable(false);
1231.                     cmbProductDetail1.setDisable(false);
1232.                     btnReturnLabelTrackingEdit1.setDisable(false);
1233.                     btnInitialEvaluationEdit1.setDisable(false);
1234.                     if (DBService.getInstance().getRole().equals("admins"))
1235.                         btnEngineeringEvaluationEdit1.setDisable(false);
1236.                     cmbDisposition1.setDisable(false);
1237.                     btnDispositionNotesEdit1.setDisable(false);
1238.                     if (!model.getSelectedCreditReplaceRepair().equals("Credit")) {
1239.                         btnReplacementRepairTrackingEdit1.setDisable(false);
1240.                         dpReplacementRepairDate1.setDisable(false);
1241.                         cbReplacementRepairShip1.setDisable(false);
1242.                     }
1243.                 } catch (SQLException e) {
1244.                     Alert a = new Alert(Alert.AlertType.ERROR);
1245.                     a.setContentText(
1246.                         "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
1247.                         e.getLocalizedMessage() + System.lineSeparator() +
1248.                         "Please click \"Save\" to try again."
1249.                     );
1250.                     a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
1251.                     a.showAndWait();
1252.                 }
1253.             }
1254.         }
1255.     }
1256.
1257.     /** btnInitialEvaluationEdit1OnAction first asks the user whether they want to edit
    the {@link TextArea}, and if so,
1258.      *  changes the txtInitialEvaluation1 {@link TextArea} to editable, the "Edit" button
    to "Save", and disables
1259.      *  all other modifiable (or used to modify) controls while the editing is in
    progress. Once the user clicks on
1260.      *  "Save", the new contents of the {@link TextArea} will be pushed to the database,
    the disabled controls will be
```

```
1261.      *  re-enabled, and the button text will revert back to "Edit", along with changing
    the {@link TextArea} back to
1262.      *  non-editable.
1263.      * @param event The {@link ActionEvent} that is passed to this event handler.
1264.      * (Requirement 5.2.1, 5.2.1.1, 5.2.1.2, 5.2.1.3)
1265.      */
1266.     @FXML
1267.     protected void btnInitialEvaluationEdit1OnAction(ActionEvent event) {
1268.         if (btnInitialEvaluationEdit1.getText().equals("Edit")) {
1269.             // Ask the user if they want to edit the field.
1270.             if (editConfirmationMessage(btnInitialEvaluationEdit1, "Initial Evaluation",
    "Save")) {
1271.                 // Set field to be editable and disable the other modifiable (or used to
    modify) controls.
1272.                 txtInitialEvaluation1.setEditable(true);
1273.                 cmbEmployeeInfo1.setDisable(true);
1274.                 cmbRMAStatus1.setDisable(true);
1275.                 cmbCreditReplaceRepair1.setDisable(true);
1276.                 cmbReasonCode1.setDisable(true);
1277.                 btnSpecialInstructionEdit1.setDisable(true);
1278.                 cmbProductDetail1.setDisable(true);
1279.                 btnReturnLabelTrackingEdit1.setDisable(true);
1280.                 btnQuantityEdit1.setDisable(true);
1281.                 btnEngineeringEvaluationEdit1.setDisable(true);
1282.                 cmbDisposition1.setDisable(true);
1283.                 btnDispositionNotesEdit1.setDisable(true);
1284.                 if (!model.getSelectedCreditReplaceRepair().equals("Credit")) {
1285.                     btnReplacementRepairTrackingEdit1.setDisable(true);
1286.                     dpReplacementRepairDate1.setDisable(true);
1287.                     cbReplacementRepairShip1.setDisable(true);
1288.                 }
1289.             }
1290.         } else {
1291.             // Ask the user if they want to save the field.
1292.             if (editConfirmationMessage(btnInitialEvaluationEdit1, "Initial Evaluation",
    "Edit")) {
1293.                 try {
1294.                     // Update the field value in the database.
1295.                     DBService.getInstance().updateRMAInitialEvaluation(model.getRmaId(),
    model.getInitialEvaluation());
1296.                     updateLastModifiedDetails();
1297.                     model.updateRMAProgress();
1298.
1299.                     // Set field to be non-editable and enable the other modifiable (or
    used to modify) controls.
1300.                     txtInitialEvaluation1.setEditable(false);
1301.                     cmbEmployeeInfo1.setDisable(false);
1302.                     cmbRMAStatus1.setDisable(false);
1303.                     cmbCreditReplaceRepair1.setDisable(false);
1304.                     cmbReasonCode1.setDisable(false);
1305.                     btnSpecialInstructionEdit1.setDisable(false);
1306.                     cmbProductDetail1.setDisable(false);
1307.                     btnReturnLabelTrackingEdit1.setDisable(false);
1308.                     btnQuantityEdit1.setDisable(false);
1309.                     if (DBService.getInstance().getRole().equals("admins"))
1310.                         btnEngineeringEvaluationEdit1.setDisable(false);
1311.                     cmbDisposition1.setDisable(false);
1312.                     btnDispositionNotesEdit1.setDisable(false);
1313.                     if (!model.getSelectedCreditReplaceRepair().equals("Credit")) {
1314.                         btnReplacementRepairTrackingEdit1.setDisable(false);
1315.                         dpReplacementRepairDate1.setDisable(false);
1316.                         cbReplacementRepairShip1.setDisable(false);
1317.                     }
1318.                 } catch (SQLException e) {
1319.                     Alert a = new Alert(Alert.AlertType.ERROR);
```

```
1320.                     a.setContentText(
1321.                         "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
1322.                         e.getLocalizedMessage() + System.lineSeparator() +
1323.                         "Please click \"Save\" to try again."
1324.                     );
1325.                     a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
1326.                     a.showAndWait();
1327.                 }
1328.             }
1329.         }
1330.     }
1331.
1332.     /** btnEngineeringEvaluationEdit1OnAction first asks the user whether they want to
    edit the {@link TextArea}, and if so,
1333.      *  changes the txtEngineeringEvaluation1 {@link TextArea} to editable, the "Edit"
    button to "Save", and disables
1334.      *  all other modifiable (or used to modify) controls while the editing is in
    progress. Once the user clicks on
1335.      *  "Save", the new contents of the {@link TextArea} will be pushed to the database,
    the disabled controls will be
1336.      *  re-enabled, and the button text will revert back to "Edit", along with changing
    the {@link TextArea} back to
1337.      *  non-editable.
1338.      * @param event The {@link ActionEvent} that is passed to this event handler.
1339.      * (Requirement 5.2.1, 5.2.1.1, 5.2.1.2, 5.2.1.3)
1340.      */
1341.     @FXML
1342.     protected void btnEngineeringEvaluationEdit1OnAction(ActionEvent event) {
1343.         if (btnEngineeringEvaluationEdit1.getText().equals("Edit")) {
1344.             // Ask the user if they want to edit the field.
1345.             if (editConfirmationMessage(btnEngineeringEvaluationEdit1, "Engineering
    Evaluation", "Save")) {
1346.                 // Set field to be editable and disable the other modifiable (or used to
    modify) controls.
1347.                 txtEngineeringEvaluation1.setEditable(true);
1348.                 if (DBService.getInstance().getRole().equals("admins")) {
1349.                     cmbEmployeeInfo1.setDisable(true);
1350.                     cmbRMAStatus1.setDisable(true);
1351.                     cmbCreditReplaceRepair1.setDisable(true);
1352.                     cmbReasonCode1.setDisable(true);
1353.                     btnSpecialInstructionEdit1.setDisable(true);
1354.                     cmbProductDetail1.setDisable(true);
1355.                     btnReturnLabelTrackingEdit1.setDisable(true);
1356.                     btnQuantityEdit1.setDisable(true);
1357.                     btnInitialEvaluationEdit1.setDisable(true);
1358.                     cmbDisposition1.setDisable(true);
1359.                     btnDispositionNotesEdit1.setDisable(true);
1360.                     if (!model.getSelectedCreditReplaceRepair().equals("Credit")) {
1361.                         btnReplacementRepairTrackingEdit1.setDisable(true);
1362.                         dpReplacementRepairDate1.setDisable(true);
1363.                         cbReplacementRepairShip1.setDisable(true);
1364.                     }
1365.                 }
1366.             }
1367.         } else {
1368.             // Ask the user if they want to save the field.
1369.             if (editConfirmationMessage(btnEngineeringEvaluationEdit1, "Engineering
    Evaluation", "Edit")) {
1370.                 try {
1371.                     // Update the field value in the database.
1372.
    DBService.getInstance().updateRMAEngineeringEvaluation(model.getRmaId(),
    model.getEngineeringEvaluation());
1373.                     updateLastModifiedDetails();
```

```
1374.                     model.updateRMAProgress();
1375.
1376.                     // Set field to be non-editable and enable the other modifiable (or
    used to modify) controls.
1377.                     txtEngineeringEvaluation1.setEditable(false);
1378.                     if (DBService.getInstance().getRole().equals("admins")) {
1379.                         cmbEmployeeInfo1.setDisable(false);
1380.                         cmbRMAStatus1.setDisable(false);
1381.                         cmbCreditReplaceRepair1.setDisable(false);
1382.                         cmbReasonCode1.setDisable(false);
1383.                         btnSpecialInstructionEdit1.setDisable(false);
1384.                         cmbProductDetail1.setDisable(false);
1385.                         btnReturnLabelTrackingEdit1.setDisable(false);
1386.                         btnQuantityEdit1.setDisable(false);
1387.                         btnInitialEvaluationEdit1.setDisable(false);
1388.                         cmbDisposition1.setDisable(false);
1389.                         btnDispositionNotesEdit1.setDisable(false);
1390.                         if (!model.getSelectedCreditReplaceRepair().equals("Credit")) {
1391.                             btnReplacementRepairTrackingEdit1.setDisable(false);
1392.                             dpReplacementRepairDate1.setDisable(false);
1393.                             cbReplacementRepairShip1.setDisable(false);
1394.                         }
1395.                     }
1396.                 } catch (SQLException e) {
1397.                     Alert a = new Alert(Alert.AlertType.ERROR);
1398.                     a.setContentText(
1399.                         "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
1400.                         e.getLocalizedMessage() + System.lineSeparator() +
1401.                         "Please click \"Save\" to try again."
1402.                     );
1403.                     a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
1404.                     a.showAndWait();
1405.                 }
1406.             }
1407.         }
1408.     }
1409.
1410.     /** btnDispositionNotesEdit1OnAction first asks the user whether they want to edit the
    {@link TextArea}, and if so,
1411.      *  changes the txtDispositionNotes1 {@link TextArea} to editable, the "Edit" button
    to "Save", and disables
1412.      *  all other modifiable (or used to modify) controls while the editing is in
    progress. Once the user clicks on
1413.      *  "Save", the new contents of the {@link TextArea} will be pushed to the database,
    the disabled controls will be
1414.      *  re-enabled, and the button text will revert back to "Edit", along with changing
    the {@link TextArea} back to
1415.      *  non-editable.
1416.      * (Requirement 5.2.1, 5.2.1.1, 5.2.1.2, 5.2.1.3)
1417.      * @param event The {@link ActionEvent} that was passed to this method.
1418.      */
1419.     @FXML
1420.     protected void btnDispositionNotesEdit1OnAction(ActionEvent event) {
1421.         if (btnDispositionNotesEdit1.getText().equals("Edit")) {
1422.             // Ask the user if they want to edit the field.
1423.             if (editConfirmationMessage(btnDispositionNotesEdit1, "Disposition Notes",
    "Save")) {
1424.                 // Set field to be editable and disable the other modifiable (or used to
    modify) controls.
1425.                 txtDispositionNotes1.setEditable(true);
1426.                 cmbEmployeeInfo1.setDisable(true);
1427.                 cmbRMAStatus1.setDisable(true);
1428.                 cmbCreditReplaceRepair1.setDisable(true);
1429.                 cmbReasonCode1.setDisable(true);
```

```
1430.                 btnSpecialInstructionEdit1.setDisable(true);
1431.                 cmbProductDetail1.setDisable(true);
1432.                 btnReturnLabelTrackingEdit1.setDisable(true);
1433.                 btnQuantityEdit1.setDisable(true);
1434.                 btnInitialEvaluationEdit1.setDisable(true);
1435.                 btnEngineeringEvaluationEdit1.setDisable(true);
1436.                 cmbDisposition1.setDisable(true);
1437.                 if (!model.getSelectedCreditReplaceRepair().equals("Credit")) {
1438.                     btnReplacementRepairTrackingEdit1.setDisable(true);
1439.                     dpReplacementRepairDate1.setDisable(true);
1440.                     cbReplacementRepairShip1.setDisable(true);
1441.                 }
1442.             }
1443.         } else {
1444.             // Ask the user if they want to save the field.
1445.             if (editConfirmationMessage(btnDispositionNotesEdit1, "Disposition Notes",
    "Edit")) {
1446.                 try {
1447.                     // Update the field value in the database.
1448.                     DBService.getInstance().updateRMADispositionNotes(model.getRmaId(),
    model.getDispositionNotes());
1449.                     updateLastModifiedDetails();
1450.                     model.updateRMAProgress();
1451.
1452.                     // Set field to be non-editable and enable the other modifiable (or
    used to modify) controls.
1453.                     txtDispositionNotes1.setEditable(false);
1454.                     cmbEmployeeInfo1.setDisable(false);
1455.                     cmbRMAStatus1.setDisable(false);
1456.                     cmbCreditReplaceRepair1.setDisable(false);
1457.                     cmbReasonCode1.setDisable(false);
1458.                     btnSpecialInstructionEdit1.setDisable(false);
1459.                     cmbProductDetail1.setDisable(false);
1460.                     btnReturnLabelTrackingEdit1.setDisable(false);
1461.                     btnQuantityEdit1.setDisable(false);
1462.                     btnInitialEvaluationEdit1.setDisable(false);
1463.                     if (DBService.getInstance().getRole().equals("admins"))
1464.                         btnEngineeringEvaluationEdit1.setDisable(false);
1465.                     cmbDisposition1.setDisable(false);
1466.                     if (!model.getSelectedCreditReplaceRepair().equals("Credit")) {
1467.                         btnReplacementRepairTrackingEdit1.setDisable(false);
1468.                         dpReplacementRepairDate1.setDisable(false);
1469.                         cbReplacementRepairShip1.setDisable(false);
1470.                     }
1471.                 } catch (SQLException e) {
1472.                     Alert a = new Alert(Alert.AlertType.ERROR);
1473.                     a.setContentText(
1474.                         "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
1475.                         e.getLocalizedMessage() + System.lineSeparator() +
1476.                         "Please click \"Save\" to try again."
1477.                     );
1478.                     a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
1479.                     a.showAndWait();
1480.                 }
1481.             }
1482.         }
1483.     }
1484.
1485.     /** btnReplacementRepairTrackingEdit1OnAction first asks the user whether they want to
    edit the {@link TextField}, and if
1486.      *  so, changes the txtReplacementRepairTracking1 {@link TextField} to editable, the
    "Edit" button to "Save", and disables all other
1487.      *  modifiable (or used to modify) controls while the editing is in progress. Once the
    user clicks on "Save", the
```

```
1488.      *  new contents of the {@link TextField} will be pushed to the database, the disabled
    controls will be re-enabled,
1489.      *  and the button text will revert back to "Edit", along with changing the {@link
    TextField} back to non-editable.
1490.      * (Requirement 5.2.1, 5.2.1.1, 5.2.1.2, 5.2.1.3)
1491.      * @param event The {@link ActionEvent} that was passed to this method.
1492.      */
1493.     @FXML
1494.     protected void btnReplacementRepairTrackingEdit1OnAction(ActionEvent event) {
1495.         if (btnReplacementRepairTrackingEdit1.getText().equals("Edit")) {
1496.             // Ask the user if they want to edit the field.
1497.             if (editConfirmationMessage(btnReplacementRepairTrackingEdit1,
    "Replacement/Repair Tracking #", "Save")) {
1498.                 // Set field to be editable and disable the other modifiable (or used to
    modify) controls.
1499.                 txtReplacementRepairTracking1.setEditable(true);
1500.                 cmbEmployeeInfo1.setDisable(true);
1501.                 cmbRMAStatus1.setDisable(true);
1502.                 cmbCreditReplaceRepair1.setDisable(true);
1503.                 cmbReasonCode1.setDisable(true);
1504.                 btnSpecialInstructionEdit1.setDisable(true);
1505.                 cmbProductDetail1.setDisable(true);
1506.                 btnReturnLabelTrackingEdit1.setDisable(true);
1507.                 btnQuantityEdit1.setDisable(true);
1508.                 btnInitialEvaluationEdit1.setDisable(true);
1509.                 btnEngineeringEvaluationEdit1.setDisable(true);
1510.                 cmbDisposition1.setDisable(true);
1511.                 btnDispositionNotesEdit1.setDisable(true);
1512.                 dpReplacementRepairDate1.setDisable(true);
1513.                 cbReplacementRepairShip1.setDisable(true);
1514.             }
1515.         } else {
1516.             // Ask the user if they want to save the field.
1517.             if (editConfirmationMessage(btnReplacementRepairTrackingEdit1,
    "Replacement/Repair Tracking #", "Edit")) {
1518.                 try {
1519.                     // Update the field value in the database.
1520.
    DBService.getInstance().updateRMAReplacementTrackingNumber(model.getRmaId(),
    model.getReplacementTrackingNumber());
1521.                     updateLastModifiedDetails();
1522.                     model.updateRMAProgress();
1523.
1524.                     // Set field to be non-editable and enable the other modifiable (or
    used to modify) controls.
1525.                     txtReplacementRepairTracking1.setEditable(false);
1526.                     cmbEmployeeInfo1.setDisable(false);
1527.                     cmbRMAStatus1.setDisable(false);
1528.                     cmbCreditReplaceRepair1.setDisable(false);
1529.                     cmbReasonCode1.setDisable(false);
1530.                     btnSpecialInstructionEdit1.setDisable(false);
1531.                     cmbProductDetail1.setDisable(false);
1532.                     btnReturnLabelTrackingEdit1.setDisable(false);
1533.                     btnQuantityEdit1.setDisable(false);
1534.                     btnInitialEvaluationEdit1.setDisable(false);
1535.                     if (DBService.getInstance().getRole().equals("admins"))
1536.                         btnEngineeringEvaluationEdit1.setDisable(false);
1537.                     cmbDisposition1.setDisable(false);
1538.                     btnDispositionNotesEdit1.setDisable(false);
1539.                     dpReplacementRepairDate1.setDisable(false);
1540.                     cbReplacementRepairShip1.setDisable(false);
1541.                 } catch (SQLException e) {
1542.                     Alert a = new Alert(Alert.AlertType.ERROR);
1543.                     a.setContentText(
```

```
1544.                           "Error while connecting to SQL Server database!" +
    System.lineSeparator() +
1545.                           e.getLocalizedMessage() + System.lineSeparator() +
1546.                           "Please click \"Save\" to try again."
1547.                   );
1548.                   a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
1549.                   a.showAndWait();
1550.               }
1551.           }
1552.       }
1553.   }
1554.
1555.   /** editConfirmationMessage displays the confirmation message when the user clicks an
    "Edit" or "Save" {@link Button}
1556.    *  on the form.
1557.    * @param btnName A reference to the {@link Button} in order to edit its text.
1558.    * @param name The {@link String} name of the button.
1559.    * @param editSave The resulting {@link String} text to set to the button if the user
    clicks "Yes".
1560.    * @return True if the confirmation message was accepted, or false if cancelled.
1561.    */
1562.   private boolean editConfirmationMessage(Button btnName, String name, String editSave)
    {
1563.       String msg = "";
1564.       if (btnName.getText().equals("Save"))
1565.           msg = "Would you like to save the changes to the ";
1566.       else
1567.           msg = "Would you like to edit the ";
1568.
1569.       ButtonType confirm = new ButtonType("Confirm", ButtonBar.ButtonData.OK_DONE); //
    Change the button context from OK to Confirm
1570.
1571.       Alert a = new Alert(Alert.AlertType.CONFIRMATION,msg + name + "?", confirm,
    ButtonType.CANCEL);
1572.       a.setTitle(editSave.equals("Save") ? "Confirm Edit" : "Confirm Save");
1573.       a.getDialogPane().setMinHeight(Region.USE_PREF_SIZE);
1574.       Optional<ButtonType> result = a.showAndWait();
1575.
1576.       if (result.isPresent() && result.get() == confirm) {
1577.           btnName.setText(editSave);
1578.           return true;
1579.       } else
1580.           return false;
1581.   }
1582.
1583.   /** updateLastModifiedDetails updates the Last Modified username and {@link
    LocalDateTime}.
1584.    * @throws SQLException If there is an error connecting to the database or with the
    SQL query
1585.    */
1586.   private void updateLastModifiedDetails() throws SQLException {
1587.       model.setLastModified(LocalDateTime.now());
1588.       model.setLastModifiedBy(DBService.getInstance().getUser());
1589.       DBService.getInstance().updateRMALastModified(model.getRmaId());
1590.       DBService.getInstance().updateRMALastModifiedBy(model.getRmaId());
1591.   }
1592. }
1593.
```

RMADetailsFormModel.java

```
1.  package RMA;
2.
3.  import javafx.beans.property.*;
4.  import javafx.collections.FXCollections;
5.  import javafx.collections.ObservableList;
6.  import javafx.scene.control.Alert;
7.  import javafx.scene.control.ButtonType;
8.  import javafx.scene.layout.Region;
9.
10. import java.sql.SQLException;
11. import java.time.LocalDate;
12. import java.time.Period;
13.
14. /** RMADetailsFormModel contains extended fields from RMAFormModel that display in
    RMADetailScreen.
15.  */
16. public class RMADetailsFormModel extends RMAFormModel {
17.     /** rmaProgress contains the {@link Double} value of the progress bar.
18.      */
19.     private DoubleProperty rmaProgress;
20.     /** age contains the int number of days this RMA has been open.
21.       */
22.     private IntegerProperty age;
23.     /** selectedOwner contains the user-selected {@link String} Owner for the RMA.
24.      */
25.     private StringProperty selectedOwner;
26.     /** owners is an {@link ObservableList} containing the list of {@link String} usernames
    in the RMA
27.      *  database for reassigning the RMA.
28.      */
29.     private ObservableList<String> owners;
30.     /** engineeringEvaluation contains the {@link String} engineer's evaluation for this
    RMA.
31.      */
32.     private StringProperty engineeringEvaluation;
33.     /** detailsWindowText contains the {@link String} value for the details {@link
    javafx.scene.control.TextArea}
34.      *  on the right-hand side.
35.      */
36.     private StringProperty detailsWindowText;
37.
38.
39.     /** Constructor that initializes fields to empty values.
40.      */
41.     public RMADetailsFormModel() {
42.         // Initialize properties.
43.         super();
44.         rmaProgress = new SimpleDoubleProperty();
45.         age = new SimpleIntegerProperty();
46.         selectedOwner = new SimpleStringProperty();
47.         owners = FXCollections.observableArrayList();
48.         engineeringEvaluation = new SimpleStringProperty();
49.         detailsWindowText = new SimpleStringProperty();
50.     }
51.
52.     /** getRMADetails fetches and displays the RMA details based on the passed-in RMA ID
53.      *  @param rmaId The user-provided RMA ID whose details we need to fetch from the
    database
54.      *  @throws SQLException When connection to database is not established
55.      */
56.     public void getRMADetails(String rmaId) throws SQLException {
```

```
57.         // First fetch an instance of DBService
58.         DBService dbs = DBService.getInstance();
59.
60.         // Now populate the fields using DBService
61.         setRmaId(rmaId);
62.         owners.setAll(dbs.getOwners());
63.         selectedOwner.set(dbs.getRMAOwner(rmaId));
64.         setCreated(dbs.getRMACreated(rmaId));
65.         setCreatedBy(dbs.getRMACreatedBy(rmaId));
66.         setLastModified(dbs.getRMALastModified(rmaId));
67.         setLastModifiedBy(dbs.getRMALastModifiedBy(rmaId));
68.         customerNamesProperty().setAll(dbs.getCustomerNames());
69.         setSelectedCustomerName(dbs.getRMACustomerName(rmaId));
70.
    businessNamesProperty().setAll(dbs.getCustomerBusinessNames(getSelectedCustomerName()));
71.         setSelectedBusinessName(dbs.getRMABusinessName(rmaId));
72.         setShippingAddress(dbs.getRMAAddress(rmaId));
73.         rmaStatusesProperty().setAll(dbs.getRMAStatuses());
74.         setSelectedRMAStatus(dbs.getRMAStatus(rmaId));
75.
    poNumbersProperty().setAll(dbs.getCustomerAddressPONumbers(getSelectedBusinessName()));
76.         setSelectedPONumber(dbs.getRMAPONumber(rmaId));
77.
    returnReasonCodesProperty().setAll(convertHashMapToArrayList(dbs.getReturnReasonCodes()));
78.         setSelectedReturnReasonCode(dbs.getRMAReturnReasonCode(rmaId));
79.         setSelectedCreditReplaceRepair(dbs.getRMACreditReplaceRepair(rmaId));
80.         setAdditionalInfo(dbs.getRMAAdditionalInfo(rmaId));
81.         productsProperty().setAll(dbs.getPurchaseOrderProducts(getSelectedPONumber()));
82.         setSelectedProduct(dbs.getRMAProduct(rmaId));
83.         setReturnQuantity(dbs.getRMAReturnQuantity(rmaId));
84.         setReturnLabelTracker(dbs.getRMAReturnLabelTracker(rmaId));
85.         setInitialEvaluation(dbs.getRMAInitialEvaluation(rmaId));
86.         setEngineeringEvaluation(dbs.getRMAEngineeringEvaluation(rmaId));
87.         dispositionsProperty().setAll(dbs.getDispositions());
88.         setSelectedDisposition(dbs.getRMADisposition(rmaId));
89.         setDispositionNotes(dbs.getRMADispositionNotes(rmaId));
90.         setReplacementTrackingNumber(dbs.getRMAReplacementTrackingNumber(rmaId));
91.         setReplacementShipDate(dbs.getRMAReplacementShipDate(rmaId));
92.         setShipReplacementRepair(dbs.getRMAShipReplacementRepair(rmaId));
93.     }
94.
95.     /** getRMAProgress returns the current double value of total progress in completing the
    RMA request, stored in the
96.      *  {@link DoubleProperty} rmaProgress field.
97.      * @return A double representing the total RMA progress.
98.      */
99.     public double getRMAProgress() {return rmaProgress.getValue();}
100.
101.      /** updateRMAProgress calculates the RMA progress stored in the rmaProgress field.
102.       *  (Requirement 5.1.1)
103.       */
104.      public void updateRMAProgress() {
105.          if (!getSelectedCreditReplaceRepair().equals("Credit")) {
106.              double progress = 0.0;
107.              if (!getSelectedCustomerName().isEmpty() &&
108.                  !getSelectedCustomerName().isBlank() &&
109.                   getSelectedBusinessName() != null &&
110.                  !getSelectedPONumber().isEmpty() &&
111.                  !getSelectedPONumber().isBlank() &&
112.                  !getSelectedOwner().isEmpty() &&
113.                  !getSelectedOwner().isBlank() &&
114.                  !getSelectedRMAStatus().isEmpty() &&
115.                  !getSelectedRMAStatus().isBlank() &&
116.                  !getSelectedCreditReplaceRepair().isEmpty() &&
117.                  !getSelectedCreditReplaceRepair().isBlank() &&
```

```
                !getSelectedReturnReasonCode().isEmpty() &&
                !getSelectedReturnReasonCode().isBlank()
            )
                progress += 20.0;
            if (getSelectedProduct() != null &&
               !getReturnLabelTracker().isBlank() &&
               !getReturnLabelTracker().isEmpty() &&
                getReturnQuantity() > 0
            )
                progress += 20.0;
            if (!getInitialEvaluation().isBlank() &&
                !getInitialEvaluation().isEmpty() &&
                !getEngineeringEvaluation().isBlank() &&
                !getEngineeringEvaluation().isEmpty()
            )
                progress += 20.0;
            if (!getSelectedDisposition().isEmpty() &&
                !getSelectedDisposition().isBlank() &&
                !getDispositionNotes().isEmpty() &&
                !getDispositionNotes().isBlank()
            )
                progress += 20.0;
            if (!getReplacementTrackingNumber().isBlank() &&
                !getReplacementTrackingNumber().isEmpty() &&
                 getReplacementShipDate() != null &&
                 getShipReplacementRepair()
            )
                progress += 20.0;
            rmaProgress.set(progress / 100.0);
        } else {
            double progress = 0.0;
            if(!getSelectedCustomerName().isEmpty() &&
                !getSelectedCustomerName().isBlank() &&
                 getSelectedBusinessName() != null &&
                !getSelectedPONumber().isEmpty() &&
                !getSelectedPONumber().isBlank() &&
                !getSelectedOwner().isEmpty() &&
                !getSelectedOwner().isBlank() &&
                !getSelectedRMAStatus().isEmpty() &&
                !getSelectedRMAStatus().isBlank() &&
                !getSelectedCreditReplaceRepair().isEmpty() &&
                !getSelectedCreditReplaceRepair().isBlank() &&
                !getSelectedReturnReasonCode().isEmpty() &&
                !getSelectedReturnReasonCode().isBlank()
            )
                progress += 25.0;
            if(getSelectedProduct() != null &&
              !getReturnLabelTracker().isBlank() &&
              !getReturnLabelTracker().isEmpty() &&
               getReturnQuantity() > 0
            )
                progress = progress + 25.0;
            if(!getInitialEvaluation().isBlank() &&
               !getInitialEvaluation().isEmpty() &&
               !getEngineeringEvaluation().isBlank() &&
               !getEngineeringEvaluation().isEmpty()
            )
                progress = progress + 25.0;
            if(!getSelectedDisposition().isEmpty() &&
               !getSelectedDisposition().isBlank() &&
               !getDispositionNotes().isEmpty() &&
               !getDispositionNotes().isBlank()
            )
                progress = progress + 25.0;
            rmaProgress.set(progress / 100.0);
```

```
183.          }
184.      }
185.
186.      /** rmaProgressProperty returns the {@link DoubleProperty} rmaProgress field storing
    the current value of
187.       *  the progress bar in the RMA details.
188.       * @return The {@link DoubleProperty} rmaProgress field.
189.       */
190.      public DoubleProperty rmaProgressProperty() {return rmaProgress;}
191.
192.      /** getAge returns the integer number of days this RMA request has been open.
193.       * @return An int containing the number of days this RNA request has been open.
194.       */
195.      public int getAge() {return age.getValue();}
196.
197.      /** updateAge calculates the value for age depending on whether the RMA request is
    still open
198.       *  (if Status is not Closed, used created field, otherwise use lastModified field).
199.       */
200.      public void updateAge() {
201.          if (!getSelectedRMAStatus().equals("Closed"))
202.              age.set(Period.between(getCreated().toLocalDate(),
    LocalDate.now()).getDays());
203.          else
204.              age.set(Period.between(getCreated().toLocalDate(),
    getLastModified().toLocalDate()).getDays());
205.      }
206.
207.      /** ageProperty returns the {@link IntegerProperty} age field.
208.       * @return the {@link IntegerProperty} age field.
209.       */
210.      public IntegerProperty ageProperty() {return age;}
211.
212.      /** getSelectedOwner returns the stored {@link String} username of the currently
    assigned owner, stored in the
213.       *  selectedOwner field.
214.       * @return The {@link String} username of the owner of this RMA stored in
    selectedOwner.
215.       */
216.      public String getSelectedOwner() {return selectedOwner.getValueSafe();}
217.
218.      /** setSelectedOwner stores the user-selected new {@link String} owner username into
    the selectedOwner field.
219.       * @param newSelectedOwner The new {@link String} owner's username to store in
    selectedOwner.
220.       */
221.      public void setSelectedOwner(String newSelectedOwner)
    {selectedOwner.set(newSelectedOwner);}
222.
223.      /** selectedOwnerProperty returns the {@link StringProperty} owner field.
224.       * @return The {@link StringProperty} owner field.
225.       */
226.      public StringProperty selectedOwnerProperty() {return selectedOwner;}
227.
228.      /** ownersProperty returns the {@link ObservableList} backing the Owner dropdown.
229.       * @return The {@link ObservableList} owners field.
230.       */
231.      public ObservableList<String> ownersProperty() {return owners;}
232.
233.      /** getEngineeringEvaluation returns the {@link String} stored in
    engineeringEvaluation.
234.       * @return The {@link String} text of the Engineering Evaluation.
235.       */
236.      public String getEngineeringEvaluation() {return
    engineeringEvaluation.getValueSafe();}
```

```
237.
238.      /** setEngineeringEvaluation sets the {@link String} Engineering Evaluation value
    stored in the
239.       *  engineeringEvaluation field to the passed-in newEngineeringEvaluation value.
240.       * @param  newEngineeringEvaluation The new {@link String} Engineering Evaluation to
    store in engineeringEvaluation.
241.       */
242.      public void setEngineeringEvaluation(String newEngineeringEvaluation)
    {engineeringEvaluation.setValue(newEngineeringEvaluation);}
243.
244.      /** engineeringEvaluationProperty returns the {@link StringProperty} backing the
    Engineering Evaluation {@link javafx.scene.control.TextArea}.
245.       * @return The {@link StringProperty} engineeringEvaluation field.
246.       */
247.      public StringProperty engineeringEvaluationProperty() {return engineeringEvaluation;}
248.
249.      /** getDetailsWindowText returns the details text {@link String} stored in the
    detailsWindowText field.
250.       * @return The {@link String} stored in detailsWindowText.
251.       */
252.      public String getDetailsWindowText() {return detailsWindowText.getValueSafe();}
253.
254.
255.      /** setDetailsWindowText sets the {@link String} value to be shown in the Details
    Window.
256.       * @param newDetailsWindowText The {@link String} value to display in this Details
    Window.
257.       */
258.      public void setDetailsWindowText(String newDetailsWindowText)
    {detailsWindowText.setValue(newDetailsWindowText);}
259.
260.      /** detailsWindowTextProperty returns the {@link StringProperty} backing the Details
    Window.
261.       * @return The {@link StringProperty} detailsWindowText field.
262.       */
263.      public StringProperty detailsWindowTextProperty() {return detailsWindowText;}
264.  }
```

Testing Documentation

Black Box Testing

Login Screen Test

Title	Login with valid Credentials
Description	The user enters correct credentials and is able to log into the application
Component Tested	Login Screen
Pre Condition	No Issue connecting to the Server and Database and the user has access
Test ID	TC_01
Tester Name	Ryan McAllister-Grum
Tested Date	4/28/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
	1	Launch the Application		Login Screen Displays	Result as expected	PASS	
1.1	2	Enter Username	Username Valid	Username is correct	Result as expected	PASS	
1.2	3	Enter Password	Password Valid	Password is correct	Result as expected	PASS	
1.4	4	Select Server Instance	Instance Valid	Valid instance displays the Dropdown List and Connect button is enabled	Result as expected	PASS	
1.5	5	Select Connect Button		Access granted and Home Screen Displays	Result as expected	PASS	

Title	Login with Invalid Password
Description	The user enters incorrect password with the valid username
Component Tested	Login Screen
Pre Condition	No Issue connecting to the Server and Database and the user has access
Test ID	TC_02
Tester Name	Ryan McAllister-Grum
Tested Date	4/28/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
	1	Launch the Application		Login Screen Displays	Result as expected	PASS	
1.1	2	Enter Username	Username Valid	Username is correct	Result as expected	PASS	
1.2	3	Enter Password	Password Invalid	Password is incorrect	Result as expected	PASS	
1.4.1	4	Select Server Instance	Instance Valid	Valid instance displays the Dropdown List and Connect button is enabled	Result as expected	PASS	
1.5.1	5	Select Connect Button		Access Denied and Pop up message displays	Result as expected	PASS	

Title	Login with Invalid Username
Description	The user enters incorrect username with the valid password
Component Tested	Login Screen
Pre Condition	No Issue connecting to the Server and Database and the user has access
Test ID	TC_03
Tester Name	Ryan McAllister-Grum
Tested Date	4/28/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
	1	Launch the Application		Login Screen Displays	Result as expected	PASS	
1.1	2	Enter Username	Username Invalid	Username is incorrect	Result as expected	PASS	
1.2	3	Enter Password	Password Valid	Password is correct	Result as expected	PASS	
1.4	4	Select Server Instance	Instance Valid	Valid instance displays the Dropdown List and Connect button is enabled	Result as expected	PASS	
1.5.1	5	Select Connect Button		Access Denied and Pop up message displays	Result as expected	PASS	

Title	Login with Valid Credential and Connection to Server Instance failed
Description	The user enters credential but there is no valid server connection
Component Tested	Login Screen
Pre Condition	Issue connecting to the server
Test ID	TC_04
Tester Name	Ryan McAllister-Grum
Tested Date	4/28/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
	1	Launch the Application		Login Screen Displays	Result as expected	PASS	
1.1	2	Enter Username	Username Valid	Username is correct	Result as expected	PASS	
1.2	3	Enter Password	Password Valid	Password is correct	Result as expected	PASS	
1.4	4	Select Server Instance	Instance Valid	Valid instance displays the Dropdown List and Connect button is enabled	Result as expected	PASS	
1.5.1	5	Select Connect Button	Connection to Server Failed	Error Connection to server displays	Result as expected	PASS	

Title	No SQL Server Browser Service(s) Listening or Accessible on Network
Description	The user starts the program, and while there may be a SQL Server Instance running, there is no SQL Server Browser service listening or not configured
Component Tested	Login Screen
Pre Condition	SQL Server Browser service not configured properly or stopped
Test ID	TC_05
Tester Name	Ryan McAllister-Grum
Tested Date	4/28/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
1.4.3	1	Launch the Application		Error stating no SQL Server Browser services found on network displays, and program exits afterwards	Result as expected	PASS	

Title	Login with both invalid credentials
Description	The user enters both username and password invalid
Component Tested	Login Screen
Pre Condition	No Issue connecting to the Server and Database
Test ID	TC_06
Tester Name	Ryan McAllister-Grum
Tested Date	4/28/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
	1	Launch the Application		Login Screen Displays	Result as expected	PASS	
1.1	2	Enter Username	Username Invalid	Username is incorrect	Result as expected	PASS	
1.2	3	Enter Password	Password Invalid	Password is incorrect	Result as expected	PASS	
1.4	4	Select Server Instance	Instance Valid	Valid instance displays the Dropdown List and Connect button is enabled	Result as expected	PASS	
1.5.1	5	Select Connect Button		Access Denied and Pop up message displays	Result as expected	PASS	

Title	Login with one of the required elements missing
Description	The user is attempting to connect with one or more of the elements missing and only after entering all required elements is the connect button enabled.
Component Tested	Login Screen
Pre Condition	No Issue connecting to the Server and Database and the user has access
Test ID	TC_07
Tester Name	Ryan McAllister-Grum
Tested Date	4/28/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
	1	Launch the Application		Login Screen Displays	Result as expected	PASS	
	2	Does not enter Username	No Input	Username is empty	Result as expected	PASS	
1.2	3	Enter Password	Password Valid	Password is correct	Result as expected	PASS	
1.4.2	4	Select Server Instance	No Input	Prompt stating that username needs to be filled in before selecting an Instance	Result as expected	PASS	
1.1.1	5	Attempts to Select Connect button		Connect button is disabled and not selectable	Result as expected	PASS	
1.1	6	Enter Username	Username Valid	Username is correct	Result as expected	PASS	
	7	Delete Password	No Input	Connect button is disabled	Result as expected	PASS	
1.4.2	8	Select Server Instance	No Input	Prompt stating that password needs to be filled in before selecting an Instance	Result as expected	PASS	
1.2.1	9	Attempts to Select Connect button		Connect button is disabled and not selectable	Result as expected	PASS	
1.2	10	Enter Password	Password Valid	Password is correct	Result as expected	PASS	
1.4	13	Select Server Instance	Instance Valid	Valid instance displays the Dropdown List and Connect button is enabled	Result as expected	PASS	
1.2.1	14	Delete Password	No Input	Connect button is disabled and not selectable	Result as expected	PASS	
1.2	15	Enter Password	Password Valid	Connect button is Enabled and Selectable	Result as expected	PASS	
1.1.1	16	Delete Username	No Input	Connect button is disabled and not selectable	Result as expected	PASS	
1.1	17	Enter Username	Username Valid	Connect button is Enabled and Selectable	Result as expected	PASS	
1.5	18	Select Connect Button		Access granted and Home Screen Displays	Result as expected	PASS	

New RMA Form Test

Title	Saved Button All Available Fields Completed
Description	The user completes the entire form and enters values in each field then saves the form. Test to ensure the proper order of selection is follows: Customer Name, Business Name, PO Number, Product.
Component Tested	RMA Form
Pre Condition	No Issue connecting to the Server and Database.
Test ID	TC_08
Tester Name	Ryan McAllister-Grum
Tested Date	4/28/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
2.4, 2.4.1	1	Select New RMA Button from RMA List View window		New RMA Form Displays	Result as expected	PASS	
4.18	2	Verify Business Name, PO Number, and Product dropdowns are not enabled.		Dropdowns are not Enabled	Result as expected	PASS	
4.1	3	Select Customer Name dropdown	"Big Food Inc"	List of Customer Name Displays	Result as expected	PASS	
4.18.1	4	Verify Selected Customer Name displays		"Big Food Inc" Displays and Business Name Enabled	Result as expected	PASS	
4.2	5	Select Business Name dropdown	"George's Corner Store"	List of Business Name Associated with Customer Displays Only	Result as expected	PASS	
4.18.2	6	Verify Selected Business Name displays		"George's Corner Store" Displays and PO Number Enabled	Result as expected	PASS	
4.12, 4.12.1	7	Verify Shipping Address auto populated	"63 S Mission Rd Eastborough Sedgwick, KS 67207"	"Big Food Inc. George's Corner Store 63 S Mission Rd Eastborough Sedgwick KS 67207 United States of America Phone: 621-187-3888" Displays	Result as expected	PASS	
4.11	8	Select P.O. Number dropdown	"3C2218QN"	List of P.O. Numbers Associated with Business Displays Only	Result as expected	PASS	
4.18.3	9	Verify Selected P.O. Number displays		"3C2218QN" Displays and Product Enabled	Result as expected	PASS	
4.3	10	Select Return Reason Code dropdown	"BROKEN-RET Damaged-Replace when returned"	List of Return Reason Code Displays	Result as expected	PASS	
	11	Verify Selected Reason Code displays		"BROKEN-RET Damaged-Replace when returned" Displays	Result as expected	PASS	
4.4	12	Select Credit, Replace, Repair? dropdown	"Repair"	List of Credit, Replace, Repair Displays	Result as expected	PASS	
	13	Verify Selected Credit, Replace, Repair? displays		"Repair" Displays	Result as expected	PASS	
4.5, 4.5.1	14	Select RMA Status Dropdown and "Close" does not Display	"Diagnose"	List of RMA Statuses Displays besides "Close"	Result as expected	PASS	
	15	Verify Selected RMA Status Displays		"Diagnose" Displays	Result as expected	PASS	
4.13	16	Enter text in the the Additional Information/Special Instruction	"This is a test, it is only a test"	Entered Text Displays "This is a test, it is only a test"	Result as expected	PASS	
4.7, 4.7.1	17	Select Product dropdown	"Food, Apple"	List of Products Displays	Result as expected	PASS	
	18	Verify Selected Product displays		"Food, Apple" Displays	Result as expected	PASS	
4.7.2	19	Enter Quantity	"5"	Entered Quantity Displays	Result as expected	PASS	
4.7.3	20	Enter Return Label Tracking number	"#TestReturn005"	Entered Return Label Tracking # Displays	Result as expected	PASS	
4.8	21	Enter Initial Evaluation	"Test Initial Evaluation Worked"	Entered Initial Evaluation Displays	Result as expected	PASS	
4.9.1	22	Select Disposition dropdown	"Repair and add to inventory, credit customer"	List of Disposition Displays	Result as expected	PASS	
	23	Verify Selected Disposition displays		"Repair and add to inventory, credit customer" Displays	Result as expected	PASS	
4.9.2	24	Enter Disposition Notes	"Test disposition"	Entered Disposition Displays	Result as expected	PASS	
4.14	25	Enter Replacement/Repair Product Tracking #	"#TestReplaceRepair005"	Entered Tracking # Displays	Result as expected	PASS	
4.15	26	Select a date from the calendar for Replacement/Repair Product Ship Date	Select today's date	Today's Date displays	Result as expected	PASS	
4.16	27	Check the Replacement/Repair Product Ship? Box		Checkbox is filled	Result as expected	PASS	
4.10, 4.10.1	28	Select Save Button		Confirmation pop up displays	Result as expected	PASS	
4.10.1	29	Confirm Button Selected		Confirmation pop up closes and New RMA is created and stored in the database. Focus is returned to NEW RMA form screen.	Result as expected	PASS	
3.0, 5.2.2	30	Verify in the Database New RMA Save Successful	Completed RMA Form	New RMA create and RMAID auto generated and saved in the database	Result as expected	PASS	
3.8,3.9	31	Verify all values selected and entered were saved correctly into the database	All RMA entered values	All values were stored into the database	Result as expected	PASS	
3.8,3.9	32	Verify RMA ID auto generated and assigned	Auto Generated unique ID	Unique RMA ID auto generated and stored	Result as expected	PASS	
5.2.2	33	Verify Owner auto assigned to Creator	Auto assigned to Owner	Owner and Creator are stored in the database as the same	Result as expected	PASS	

Title	Save & Close Button All Available Fields Completed
Description	The user completes the entire form and enters values in each field then saves then close the form.
Component Tested	RMA Form
Pre Condition	No Issue connecting to the Server and Database.
Test ID	TC_09
Tester Name	Ryan McAllister-Grum
Tested Date	4/28/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
2.4, 2.4.1	1	Select New RMA Button from RMA List View window		New RMA Form displays	Result as expected	PASS	
4.18	2	Verify Business Name, PO Number, and Product dropdowns are not enabled.		Dropdowns are not Enabled	Result as expected	PASS	
4.1	3	Select Customer Name dropdown	"Electronia Inc."	List of Customer Name Displays	Result as expected	PASS	
4.18.1	4	Verify Selected Customer Name displays		"Electronia Inc." Displays and Business Name Enabled	Result as expected	PASS	
4.2	5	Select Business Name dropdown	"Fried Electronics"	List of Business Name Associated with Customer Displays Only	Result as expected	PASS	
4.18.2	6	Verify Selected Business Name displays		"Fried Electronics" Displays and PO Number is enabled	Result as expected	PASS	
4.12, 4.12.1	7	Verify Shipping Address auto populated	"Electronia Inc. Fried Electronics 12 S Lander St Seattle King WA 98134 United States of America Phone: 144-783-5468 Fax: 135-874-5577"	"Electronia Inc. Fried Electronics 12 S Lander St Seattle King WA 98134 United States of America Phone: 144-783-5468 Fax: 135-874-5577" Displays	Result as expected	PASS	
4.11	8	Select P.O. Number dropdown	"48593FF29"	List of P.O. Numbers Associated with Business Displays Only	Result as expected	PASS	
4.18.3	9	Verify Selected P.O. Number displays		"48593FF29" Displays and Product is Enabled	Result as expected	PASS	

4.3	10	Select Return Reason Code dropdown	"BROKEN-FLD Scrapped by Customer"	List of Return Reason Code Displays	Result as expected	PASS	
	11	Verify Selected Reason Code displays		"BROKEN-FLD Scrapped by Customer" Displays	Result as expected	PASS	
4.4	12	Select Credit, Replace, Repair? dropdown	"Replace"	List of Credit, Replace, Repair Displays	Result as expected	PASS	
	13	Verify Selected Credit, Replace, Repair? displays		"Replace" Displays	Result as expected	PASS	
4.5, 4.5.1	14	Select RMA Status Dropdown and "Close" does not Display	"Pulled with Tech"	List of RMA Statuses Displays besides "Closed"	Result as expected	PASS	
	15	Verify Selected RMA Status Displays		"Pulled with Tech" Displays	Result as expected	PASS	
4.13	16	Enter text in the the Additional Information/Special Instruction	"This is a test, it is only a test"	Entered Text Displays "This is a test, it is only a test"	Result as expected	PASS	
4.7, 4.7.1	17	Select Product dropdown	"Electronics, Speaker"	List of Produst Displays	Result as expected	PASS	
	18	Verify Selected Product displays		"Electronics, Speaker" Displays	Result as expected	PASS	
4.7.2	19	Enter Quantity	"2"	Entered Quantity Displays	Result as expected	PASS	
4.7.3	20	Enter Return Label Tracking number	"#TestReturn002"	Entered Return Label Tracking # Displays	Result as expected	PASS	
4.8	21	Enter Initial Evaluation	"Test Initial Evaluation Worked"	Entered Initial Evaluation Displays	Result as expected	PASS	
4.9.1	22	Select Disposition dropdown	"Scrap item, credit customer"	List of Disposition Displays	Result as expected	PASS	
	23	Verify Selected Disposition displays		"Scrap item, credit customer" Displays	Result as expected	PASS	
4.9.2	24	Enter Disposition Notes	"Test disposition"	Entered Disposition Displays	Result as expected	PASS	
4.14	25	Enter Replacement/Repair Product Tracking #	"#TestReplaceRepair002"	Entered Tracking # Displays	Result as expected	PASS	
4.15	26	Select a date from the calendar for Replacement/Repair Product Ship Date	Select today's date	Today's Date displays	Result as expected	PASS	
4.16	27	Check the Replacement/Repair Product Ship? Box		Checkbox is filled	Result as expected	PASS	
4.10, 4.10.1	28	Save & Close is selected. No Issue connecting to the Server and Database and the user has access. User logged in is "admin"		Confirmation pop up displays	Result as expected	PASS	
4.10.1	29	Confirm Button Selected		Confirmation pop up closes and New RMA is created and stored in the database. User is returned to the Home Screen.	Result as expected	PASS	
2.5.1	30			Table view is refreshed and new RMA saved displays	Result as expected	PASS	
3.0, 5.2.2	31	Verify in the Database New RMA Save Successful	Completed RMA Form	New RMA create and RMAID auto generated and saved in the database	Result as expected	PASS	
3.8,3.9	32	Verify all values selected and entered were saved correctly into the database	All RMA entered values	All values were stored into the database	Result as expected	PASS	
3.8,3.9	33	Verify RMA ID auto generated and assigned	Auto Generated unique ID	Unique RMA ID auto generated and stored	Result as expected	PASS	
5.2.2	34	Verify Owner auto assigned to Creator	Auto assigned to Owner	Owner and Creator are stored in the database as the same	Result as expected	PASS	

Title	**Cancel Button All Available Fields Completed**
Description	The user cancels and no new RMA were saved.
Component Tested	RMA Form
Pre Condition	No Issue connecting to the Server and Database.
Test ID	TC_10
Tester Name	Ryan McAllister-Grum
Tested Date	4/28/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
2.4, 2.4.1	1	Select New RMA Button from RMA List View window		New RMA Form displays	Result as expected	PASS	
4.18	2	Verify Business Name, PO Number, and Product dropdowns are not enabled.		Dropdowns are not Enabled	Result as expected	PASS	
4.1	3	Select Customer Name dropdown	"Everything Inc."	List of Customer Name Displays	Result as expected	PASS	
4.18.1	4	Verify Selected Customer Name displays		"Everything Inc." Displays and Business Name is Enabled	Result as expected	PASS	
4.2	5	Select Business Name dropdown	"Open Box Mart"	List of Business Name Associated with Customer Displays Only	Result as expected	PASS	
4.18.2	6	Verify Selected Business Name displays		"Open Box Mart" Displays and PO Number is Enabled	Result as expected	PASS	
4.12, 4.12.1	7	Verify Shipping Address auto populated	"Everything Inc. Open Box Mart 100 W 100 S 200 N 350 E Redmond Sevier UT 84652 United States of America Phone: 122-345-6789"	"Everything Inc. Open Box Mart 100 W 100 S 200 N 350 E Redmond Sevier UT 84652 United States of America Phone: 122-345-6789" Displays	Result as expected	PASS	
4.11	8	Select P.O. Number dropdown	"693823V342"	List of P.O. Numbers Associated with Business Displays Only	Result as expected	PASS	
4.18.3	9	Verify Selected P.O. Number displays		"693823V342" Displays and Product is Enabled	Result as expected	PASS	
4.3	10	Select Return Reason Code dropdown	"ERROR Customer ordered wrong item"	List of Return Reason Code Displays	Result as expected	PASS	
	11	Verify Selected Reason Code displays		"ERROR Customer ordered wrong item" Displays	Result as expected	PASS	
4.4	12	Select Credit, Replace, Repair? dropdown	"Credit"	List of Credit, Replace, Repair Displays	Result as expected	PASS	
	13	Verify Selected Credit, Replace, Repair? displays		"Credit" Displays	Result as expected	PASS	
4.5, 4.5.1	14	Select RMA Status Dropdown and "Close" does not Display	"Diagnose"	List of RMA Statuses Displays besides "Closed"	Result as expected	PASS	
	15	Verify Selected RMA Status Displays		"Diagnose" Displays	Result as expected	PASS	
4.13	16	Enter text in the the Additional Information/Special Instruction	"This is a test, it is only a test"	Entered Text Displays "This is a test, it is only a test"	Result as expected	PASS	
4.7, 4.7.1	17	Select Product dropdown	"Car Parts, Engine"	List of Produst Displays	Result as expected	PASS	
	18	Verify Selected Product displays		"Car Parts, Engine" Displays	Result as expected	PASS	
4.7.2	19	Enter Quantity	"8"	Entered Quantity Displays	Result as expected	PASS	
4.7.3	20	Enter Return Label Tracking number	"#TestReturn005"	Entered Return Label Tracking # Displays	Result as expected	PASS	
4.8	21	Enter Initial Evaluation	"Test Initial Evaluation Worked"	Entered Initial Evaluation Displays	Result as expected	PASS	
4.9.1	22	Select Disposition dropdown	"Repair and add to inventory, credit customer"	List of Disposition Displays	Result as expected	PASS	
	23	Verify Selected Disposition displays		"Repair and add to inventory, credit customer" Displays	Result as expected	PASS	
4.9.2	24	Enter Disposition Notes	"Test disposition"	Entered Disposition Displays	Result as expected	PASS	
5.2.7	25	Verify All fields in Replacement Information are disabled.		All Replacement Information Fields are Not Enabled	Result as expected	PASS	
4.10, 4.10.3	26	Select Cancel Button		Confirmation pop up displays	Result as expected	PASS	
	27	Cancel Button Selected	Cancel	Confirmation pop up closes and focus is returned to NEW RMA form screen. No change happened	Result as expected	PASS	
4.10, 4.10.3	28	Select Cancel Button		Confirmation pop up displays	Result as expected	PASS	
	29	Confirm button Selected	Confirm	Confirmation pop up closes and user is returned to the home screen.	Result as expected	PASS	
	30	Verify RMA was not created and added to the Table List		RMA was created and is not on the Table List	Result as expected	PASS	
2.4, 2.4.1	31	Select New RMA Button from RMA List View window		New RMA Form displays	Result as expected	PASS	
	32	Verifyl all fields are cleared and returned to default		RMA Form is cleared of all values	Result as expected	PASS	
	33	Verify RMA was not saved in the database		RMA was not saved in the database	Result as expected	PASS	

Title	**Save Buttons with Missing Required Fields**

Description	The user attempts to save with at least one missing required fields.
Component Tested	RMA Form
Pre Condition	No Issue connecting to the Server and Database.
Test ID	TC_11
Tester Name	Ryan McAllister-Grum
Tested Date	4/28/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
2.4, 2.4.1	1	Select New RMA Button from RMA List View window		New RMA Form displays	Result as expected	PASS	
4.18	2	Verify Business Name, PO Number, and Product dropdowns are not enabled.		Dropdowns are not Enabled	Result as expected	PASS	
4.1	3	Select Customer Name dropdown	"Everything Inc."	List of Customer Name Displays	Result as expected	PASS	
4.18.1	4	Verify Selected Customer Name displays		"Everything Inc." Displays and Business Name is Enabled	Result as expected	PASS	
4.10.2	5	Select Save Button		Error Pop up Missing Fields displays	Result as expected	PASS	
4.10.2	6	OK Button Selected		Error Message Closes and user is returned to New Form	Result as expected	PASS	
4.2	7	Select Business Name dropdown	"Open Box Mart"	List of Business Name Associated with Customer Displays Only	Result as expected	PASS	
4.18.2	8	Verify Selected Business Name displays		"Open Box Mart" Displays and PO Number is Enabled	Result as expected	PASS	
4.12, 4.12.1	9	Verify Shipping Address auto populated	"Everything Inc. Open Box Mart 100 W 100 S 200 N 350 E Redmond Sevier UT 84652 United States of America Phone: 122-345-6789"	"Everything Inc. Open Box Mart 100 W 100 S 200 N 350 E Redmond Sevier UT 84652 United States of America Phone: 122-345-6789" Displays	Result as expected	PASS	
4.10.2	10	Select Save and Close Button		Error Pop up Missing Fields displays	Result as expected	PASS	
4.10.2	11	OK Button Selected		Error Message Closes and user is returned to New Form	Result as expected	PASS	
4.11	12	Select P.O. Number dropdown	"693823V342"	List of P.O. Numbers Associated with Business Displays Only	Result as expected	PASS	
4.18.3	13	Verify Selected P.O. Number displays		"693823V342" Displays and Product is Enabled	Result as expected	PASS	
4.10.2	14	Select Save and Close Button		Error Pop up Missing Fields displays	Result as expected	PASS	
4.10.2	15	OK Button Selected		Error Message Closes and user is returned to New Form	Result as expected	PASS	
4.3	16	Select Return Reason Code dropdown	"ERROR Customer ordered wrong item"	List of Return Reason Code Displays	Result as expected	PASS	
	17	Verify Selected Reason Code displays		"ERROR Customer ordered wrong item" Displays	Result as expected	PASS	
4.10.2	18	Select Save and Close Button		Error Pop up Missing Fields displays	Result as expected	PASS	
4.10.2	19	OK Button Selected		Error Message Closes and user is returned to New Form	Result as expected	PASS	
4.4	20	Select Credit, Replace, Repair? dropdown	"Credit"	List of Credit, Replace, Repair Displays	Result as expected	PASS	
	21	Verify Selected Credit, Replace, Repair? displays		"Credit" Displays	Result as expected	PASS	
5.2.7	22	Verify All fields in Replacement Information are disabled.		All Replacement Information Fields are Not Enabled	Result as expected	PASS	
4.10.2	23	Select Save and Close Button		Error Pop up Missing Fields displays	Result as expected	PASS	
4.10.2	24	OK Button Selected		Error Message Closes and user is returned to New Form	Result as expected	PASS	
4.4	25	Select Credit, Replace, Repair? dropdown	"Repair"	List of Credit, Replace, Repair Displays	Result as expected	PASS	
	26	Verify Selected Credit, Replace, Repair? displays		"Repair" Displays	Result as expected	PASS	
5.2.7	27	Verify All fields in Replacement Information are enabled.		All Replacement Information Fields are Enabled	Result as expected	PASS	
4.5, 4.5.1	28	Select RMA Status Dropdown and "Closed" does not Display	"Diagnose"	List of RMA Statuses Displays besides "Closed"	Result as expected	PASS	
	29	Verify Selected RMA Status Displays		"Diagnose" Displays	Result as expected	PASS	
4.10.2	30	Select Save and Close Button		Error Pop up Missing Fields displays	Result as expected	PASS	
4.10.2	31	OK Button Selected		Error Message Closes and user is returned to New Form	Result as expected	PASS	
4.13	32	Enter text in the the Additional Information/Special Instruction	"This is a test, it is only a test"	Entered Text Displays "This is a test, it is only a test"	Result as expected	PASS	
4.7, 4.7.1	33	Select Product dropdown	"Car Parts, Engine"	List of Produst Displays	Result as expected	PASS	
	34	Verify Selected Product displays		"Car Parts, Engine" Displays	Result as expected	PASS	
4.10.2	35	Select Save Button		Error Pop up Missing Fields displays	Result as expected	PASS	
4.10.2	36	OK Button Selected		Error Message Closes and user is returned to New Form	Result as expected	PASS	
4.7.2	37	Enter Quantity	"8"	Entered Quantity Displays	Result as expected	PASS	
4.7.3	38	Enter Return Label Tracking number	"#TestReturn005"	Entered Return Label Tracking # Displays	Result as expected	PASS	
4.10, 4.10.1	39	Select Save Button		Confirmation pop up displays	Result as expected	PASS	
4.10.1	40	Confirm Button Selected		Confirmation pop up closes and New RMA is created and stored in the database. Focus is returned to NEW RMA form screen.	Result as expected	PASS	
3.0, 5.2.2	41	Verify in the Database New RMA Save Successful	Completed RMA Form	New RMA create and RMAID auto generated and saved in the database	Result as expected	PASS	
3.8,3.9	42	Verify all values selected and entered were saved correctly into the database	All RMA entered values	All values were stored into the database	Result as expected	PASS	
3.8,3.9	43	Verify RMA ID auto generated and assigned	Auto Generated unique ID	Unique RMA ID auto generated and stored	Result as expected	PASS	
5.2.2	44	Verify Owner auto assigned to Creator	Auto assigned to Owner	Owner and Creator are stored in the database as the same	Result as expected	PASS	

Title	**Save and Close with only Non-Required fields Missing**
Description	The user attempts to save with at least one non required fields missing
Component Tested	RMA Form
Pre Condition	No Issue connecting to the Server and Database.
Test ID	TC_12
Tester Name	Ryan McAllister-Grum
Tested Date	4/28/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
2.4, 2.4.1	1	Select New RMA Button from RMA List View window		New RMA Form displays	Result as expected	PASS	
4.18	2	Verify Business Name, PO Number, and Product dropdowns are not enabled.		Dropdowns are not Enabled	Result as expected	PASS	
4.1	3	Select Customer Name dropdown	"Everything Inc."	List of Customer Name Displays	Result as expected	PASS	
4.18.1	4	Verify Selected Customer Name displays		"Everything Inc." Displays and Business Name is Enabled	Result as expected	PASS	
4.2	5	Select Business Name dropdown	"Open Box Mart"	List of Business Name Associated with Customer Displays Only	Result as expected	PASS	
4.18.2	6	Verify Selected Business Name displays		"Open Box Mart" Displays and PO Number is Enabled	Result as expected	PASS	
4.12, 4.12.1	7	Verify Shipping Address auto populated	"Everything Inc. Open Box Mart 100 W 100 S 200 N 350 E Redmond Sevier UT 84652 United States of America Phone: 122-345-6789"	"Everything Inc. Open Box Mart 100 W 100 S 200 N 350 E Redmond Sevier UT 84652 United States of America Phone: 122-345-6789" Displays	Result as expected	PASS	

4.11	8	Select P.O. Number dropdown	"693823V342"	List of P.O. Numbers Associated with Business Displays Only	Result as expected	PASS	
4.18.3	9	Verify Selected P.O. Number displays		"693823V342" Displays and Product is Enabled	Result as expected	PASS	
4.3	10	Select Return Reason Code dropdown	"ERROR Customer ordered wrong item"	List of Return Reason Code Displays	Result as expected	PASS	
	11	Verify Selected Reason Code displays		"ERROR Customer ordered wrong item" Displays	Result as expected	PASS	
4.4	12	Select Credit, Replace, Repair? dropdown	"Credit"	List of Credit, Replace, Repair Displays	Result as expected	PASS	
	13	Verify Selected Credit, Replace, Repair? displays		"Credit" Displays	Result as expected	PASS	
4.5, 4.5.1	14	Select RMA Status Dropdown and "Closed" does not Display	"Diagnose"	List of RMA Statuses Displays besides "Closed"	Result as expected	PASS	
	15	Verify Selected RMA Status Displays		"Diagnose" Displays	Result as expected	PASS	
4.13	16	Enter text in the the Additional Information/Special Instruction	"This is a test, it is only a test"	Entered Text Displays "This is a test, it is only a test"	Result as expected	PASS	
4.7, 4.7.1	17	Select Product dropdown	"Car Parts, Engine"	List of Produst Displays	Result as expected	PASS	
	18	Verify Selected Product displays		"Car Parts, Engine" Displays	Result as expected	PASS	
4.7.2	19	Enter Quantity	"8"	Entered Quantity Displays	Result as expected	PASS	
4.7.3	20	Enter Return Label Tracking number	"#TestReturn005"	Entered Return Label Tracking # Displays	Result as expected	PASS	
4.10, 4.10.1	21	Select Save & Close Button		Confirmation pop up displays	Result as expected	PASS	
4.10.1	22	Confirm Button Selected		Confirmation pop up closes and New RMA is created and stored in the database. User is returned to the Home Screen.	Result as expected	PASS	
2.5.1	23			Table view is refreshed and new saved RMA displays	Result as expected	PASS	
3.0, 5.2.2	24	Verify in the Database New RMA Save Successful	Completed RMA Form	New RMA create and RMAID auto generated and saved in the database	Result as expected	PASS	
3.8,3.9	25	Verify all values selected and entered were saved correctly into the database	All RMA entered values	All values were stored into the database	Result as expected	PASS	
3.8,3.9	26	Verify RMA ID auto generated and assigned	Auto Generated unique ID	Unique RMA ID auto generated and stored	Result as expected	PASS	
5.2.2	27	Verify Owner auto assigned to Creator	Auto assigned to Owner	Owner and Creator are stored in the database as the same	Result as expected	PASS	

Title	**Cancel Button All Available Fields Completed**
Description	The user cancels and no new RMA were saved.
Component Tested	RMA Form
Pre Condition	No Issue connecting to the Server and Database.
Test ID	TC_13
Tester Name	Ryan McAllister-Grum
Tested Date	4/28/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
2.4, 2.4.1	1	Select New RMA Button from RMA List View window		New RMA Form displays	Result as expected	PASS	
4.18	2	Verify Business Name, PO Number, and Product dropdowns are not enabled.		Dropdowns are not Enabled	Result as expected	PASS	
4.1	3	Select Customer Name dropdown	"Everything Inc."	List of Customer Name Displays	Result as expected	PASS	
4.18.1	4	Verify Selected Customer Name displays		"Everything Inc." Displays and Business Name is Enabled	Result as expected	PASS	
4.2	5	Select Business Name dropdown	"Open Box Mart"	List of Business Name Associated with Customer Displays Only	Result as expected	PASS	
4.18.2	6	Verify Selected Business Name displays		"Open Box Mart" Displays and PO Number is Enabled	Result as expected	PASS	
4.12, 4.12.1	7	Verify Shipping Address auto populated	"Everything Inc. Open Box Mart 100 W 100 S 200 N 350 E Redmond Sevier UT 84652 United States of America Phone: 122-345-6789"	"Everything Inc. Open Box Mart 100 W 100 S 200 N 350 E Redmond Sevier UT 84652 United States of America Phone: 122-345-6789" Displays	Result as expected	PASS	
4.11	8	Select P.O. Number dropdown	"693823V342"	List of P.O. Numbers Associated with Business Displays Only	Result as expected	PASS	
4.18.3	9	Verify Selected P.O. Number displays		"693823V342" Displays and Product is Enabled	Result as expected	PASS	
4.3	10	Select Return Reason Code dropdown	"ERROR Customer ordered wrong item"	List of Return Reason Code Displays	Result as expected	PASS	
	11	Verify Selected Reason Code displays		"ERROR Customer ordered wrong item" Displays	Result as expected	PASS	
4.4	12	Select Credit, Replace, Repair? dropdown	"Credit"	List of Credit, Replace, Repair Displays	Result as expected	PASS	
	13	Verify Selected Credit, Replace, Repair? displays		"Credit" Displays	Result as expected	PASS	
4.5, 4.5.1	14	Select RMA Status Dropdown and "Closed" does not Display	"Diagnose"	List of RMA Statuses Displays besides "Closed"	Result as expected	PASS	
	15	Verify Selected RMA Status Displays		"Diagnose" Displays	Result as expected	PASS	
4.13	16	Enter text in the the Additional Information/Special Instruction	"This is a test, it is only a test"	Entered Text Displays "This is a test, it is only a test"	Result as expected	PASS	
4.7, 4.7.1	17	Select Product dropdown	"Car Parts, Engine"	List of Produst Displays	Result as expected	PASS	
	18	Verify Selected Product displays		"Car Parts, Engine" Displays	Result as expected	PASS	
4.7.2	19	Enter Quantity	"8"	Entered Quantity Displays	Result as expected	PASS	
4.7.3	20	Enter Return Label Tracking number	"#TestReturn005"	Entered Return Label Tracking # Displays	Result as expected	PASS	
4.8	21	Enter Initial Evaluation	"Test Initial Evaluation Worked"	Entered Initial Evaluation Displays	Result as expected	PASS	
4.9.1	22	Select Disposition dropdown	"Repair and add to inventory, credit customer"	List of Disposition Displays	Result as expected	PASS	
	23	Verify Selected Disposition displays		"Repair and add to inventory, credit customer" Displays	Result as expected	PASS	
4.9.2	24	Enter Disposition Notes	"Test disposition"	Entered Disposition Displays	Result as expected	PASS	
5.2.7	25	Verify All fields in Replacement Information are disabled.		All Replacement Information Fields are Not Enabled	Result as expected	PASS	
4.17, 4.10.3	26	Select Cancel Button		Confirmation pop up displays	Result as expected	PASS	
	27	Cancel Button Selected	Cancel	Confirmation pop up closes and focus is returned to NEW RMA form screen. No change happened	Result as expected	PASS	
4.17, 4.10.3	28	Select Cancel Button		Confirmation pop up displays	Result as expected	PASS	
	29	Confirm button Selected	Confirm	Confirmation pop up closes and user is returned to the home screen.	Result as expected	PASS	
	30	Verify RMA was not created and added to the Table List		RMA was not created and is not on the Table List	Result as expected	PASS	
2.4, 2.4.1	31	Select New RMA Button from RMA List View window		New RMA Form displays	Result as expected	PASS	
	32	Verify all fields are cleared and returned to default		RMA Form is cleared of all values	Result as expected	PASS	
	33	Verify RMA was not saved in the database		RMA was not saved in the database	Result as expected	PASS	

Title	**Change Made to Customer Information Field Clears Input in Business, Shipping Address, PO Number, and Product, Change Made to Business Name Clears Input in PO Number and Product, and Change Made to PO Number Clears Input in Product**
Description	The user selects a different Customer and the lower-level ComboBoxes (Business, PO Number, and Product), along with Shipping Address, are cleared.
Component Tested	RMA Form
Pre Condition	No Issue connecting to the Server and Database.
Test ID	TC_14
Tester Name	Ryan McAllister-Grum

Tested Date	4/29/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
2.4, 2.4.1	1	Select New RMA Button from RMA List View window		New RMA Form displays	Result as expected	PASS	
4.18	2	Verify Business Name, PO Number, and Product dropdowns are not enabled.		Dropdowns are not Enabled	Result as expected	PASS	
4.1	3	Select Customer Name dropdown	"Everything Inc."	List of Customer Name Displays	Result as expected	PASS	
4.18.1	4	Verify Selected Customer Name displays		"Everything Inc." Displays and Business is Enabled	Result as expected	PASS	
4.2	5	Select Business Name dropdown	"Open Box Mart"	List of Business Name Associated with Customer Displays Only	Result as expected	PASS	
4.18.2	6	Verify Selected Business Name displays		"Open Box Mart" Displays and PO Number is Enabled	Result as expected	PASS	
4.12, 4.12.1	7	Verify Shipping Address auto populated	"Everything Inc. Open Box Mart 100 W 100 S 200 N 350 E Redmond Sevier UT 84652 United States of America Phone: 122-345-6789"	"Everything Inc. Open Box Mart 100 W 100 S 200 N 350 E Redmond Sevier UT 84652 United States of America Phone: 122-345-6789" Displays	Result as expected	PASS	
4.11	8	Select P.O. Number dropdown	"693823V342"	List of P.O. Numbers Associated with Business Displays Only	Result as expected	PASS	
4.18.3	9	Verify Selected P.O. Number displays		"693823V342" Displays and Product is Enabled	Result as expected	PASS	
4.3	10	Select Return Reason Code dropdown	"ERROR Customer ordered wrong item"	List of Return Reason Code Displays	Result as expected	PASS	
	11	Verify Selected Reason Code displays		"ERROR Customer ordered wrong item" Displays	Result as expected	PASS	
4.4	12	Select Credit, Replace, Repair? dropdown	"Credit"	List of Credit, Replace, Repair Displays	Result as expected	PASS	
	13	Verify Selected Credit, Replace, Repair? displays		"Credit" Displays	Result as expected	PASS	
4.5, 4.5.1	14	Select RMA Status Dropdown and "Closed" does not Display	"Diagnose"	List of RMA Statuses Displays besides "Closed"	Result as expected	PASS	
	15	Verify Selected RMA Status Displays		"Diagnose" Displays	Result as expected	PASS	
4.13	16	Enter text in the the Additional Information/Special Instruction	"This is a test, it is only a test"	Entered Text Displays "This is a test, it is only a test"	Result as expected	PASS	
4.7, 4.7.1	17	Select Product dropdown	"Car Parts, Engine"	List of Produst Displays	Result as expected	PASS	
	18	Verify Selected Product displays		"Car Parts, Engine" Displays	Result as expected	PASS	
4.7.2	19	Enter Quantity	"8"	Entered Quantity Displays	Result as expected	PASS	
	20	Select Customer Name dropdown	"Electronia Inc."	List of Customer Name Displays	Result as expected	PASS	
	21	Select Confirm button	Confirm	Prompt display notifying user that changing the selected Customer will clear Business Name, PO Number, Shipping Address, and Product	Result as expected	PASS	
4.18.4	22	Verify Selected Customer Name displays and Business Name, PO Number, Shipping Address, and Product are empty. PO Number and Product are disabled.		"Electronia Inc." Displays and Business Name, PO Number, Shipping Address, and Product are empty. PO Number and Product are disabled.	Result as expected	PASS	
4.2	23	Select Business Name dropdown	"Fried Electronics"	List of Business Name Associated with Customer Displays Only	Result as expected	PASS	
4.18.2	24	Verify Selected Business Name displays		"Fried Electronics" Displays and PO Number is Enabled	Result as expected	PASS	
4.12, 4.12.1	25	Verify Shipping Address auto populated	"Electronia Inc. Fried Electronics 12 S Lander St Seattle King WA 98134 United States of America Phone: 144-783-5468 Fax: 135-874-5577"	"Electronia Inc. Fried Electronics 12 S Lander St Seattle King WA 98134 United States of America Phone: 144-783-5468 Fax: 135-874-5577" Displays	Result as expected	PASS	
4.11	26	Select P.O. Number dropdown	"48593FF29"	List of P.O. Numbers Associated with Business Displays Only	Result as expected	PASS	
4.18.3	27	Verify Selected P.O. Number displays		"48593FF29" Displays and Product is Enabled	Result as expected	PASS	
4.7, 4.7.1	28	Select Product dropdown	"Electronics, Speaker"	List of Product Displays	Result as expected	PASS	
	29	Verify Selected Product displays		"Electronics, Speaker" Displays	Result as expected	PASS	
4.2	30	Select Business Name dropdown	"Solid State Store"	List of Business Name Displays			
	31	Select Confirm button	Confirm	Prompt display notifying user that changing the selected Business will clear PO Number and Product	Result as expected	PASS	
4.18.4	32	Verify Selected Business Name displays and PO Number, Product are empty and Product is disabled. Shipping address is auto populated corresponding to the business name	"Electronia Inc. Solid State Store 250 Industrial Park St Building #6 Pittsfield Somerset ME 04967 United States of America Phone: 207-689-7885"	"Solid State Store" Displays, PO Number and Product are empty, and Product is disabled. "Electronia Inc. Solid State Store 250 Industrial Park St Building #6 Pittsfield Somerset ME 04967 United States of America Phone: 207-689-7885" in the Shipping Address Field Displays	Result as expected	PASS	
4.11	33	Select P.O. Number dropdown	"219389VF3"	List of P.O. Numbers Associated with Business Displays Only	Result as expected	PASS	
4.18.3	34	Verify Selected P.O. Number displays		"219389VF3" Displays and Product is Enabled	Result as expected	PASS	
4.7, 4.7.1	35	Select Product dropdown	"Electronics, Monitor"	List of Product Displays	Result as expected	PASS	
	36	Verify Selected Product displays		"Electronics, Monitor" Displays	Result as expected	PASS	
	37	Select PO Number dropdown	"9483945FGSHI"	List of PO Number Displays	Result as expected	PASS	
	38	Select Confirm button	Confirm	Prompt display notifying user that changing the selected PO Number will clear Product	Result as expected	PASS	
4.18.4	39	Verify Selected PO Number displays and Product is empty.		"9483945FGSHI" Displays and Product is empty.	Result as expected	PASS	
4.7, 4.7.1	40	Select Product dropdown	"Electronics, Speaker"	List of Produst Displays	Result as expected	PASS	
	41	Verify Selected Product displays		"Electronics, Speaker" Displays	Result as expected	PASS	

RMA Details Screen Test

Title	Edit Buttons and Verify changes Saved
Description	The user Utilizes the Edit Button and makes changes. Form is saved with every change made. Progress bar is update depending on completion of each section.
Component Tested	RMA Detail Screen
Pre Condition	No Issue connecting to the Server and Database. Previously created RMA exists and user role is "analysts".
Test ID	TC_15
Tester Name	Ryan McAllister-Grum
Tested Date	4/29/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
2.2.2	1	Select RMA Number from Table View	"0"	RMA Details Screen Displays	Result as expected	PASS	
5.0	2	Verify all values selected and entered that were previously saved matches the details screen	Saved RMA Details	All values that were stored into the database Displays correctly	Result as expected	PASS	
5.2.2	3	Verify RMA ID assigned under RMA # Label		Assigned RMA # Displays Correctly	Result as expected	PASS	
5.1, 5.1.1	4	Verify Progress Bar indicates correct percentage of completion.	Big Food Inc Previous Saved RMA	Progress Bar Displays 80%	Result as expected	PASS	
5.4, 5.4.1	5	Verify Details Area displays Customer information as default.		Customer's Information is Diplayed in the Details Area.	Result as expected	PASS	
5.2.4	6	Verify Age in Days is the difference between today's and created date	Created Date of RMA	Correct Age Displays	Result as expected	PASS	
5.2.5	7	Verify Created by and date	"analyst" and Created Date	"analyst" and Created Date Displays	Result as expected	PASS	
5.2.3	8	Verify Last Modified by and date	"analyst" and Creation Date	"analyst" and Same Date as Created Date Displays	Result as expected	PASS	
5.0, 5.8	9	Verify all values selected and entered that were previously saved matches the details screen	Saved RMA Details	All values that were stored into the database Displays correctly	Result as expected	PASS	
4.6	10	Verify the Default Owner is set to the Creator of the RMA	"analyst"	RMA Owner "analyst" Displays	Result as expected	PASS	
	11	Select RMA Owner Dropdown		List of Employee name Displays	Result as expected	PASS	
5.2.6, 5.8	12	Verify Selected Owner Displays and RMA details in the database updated accordingly	"admin"	"admin" Displays in the field and RMA Details updated in the database	Result as expected	PASS	
5.2.3	13	Verify Last Modified by and date	"analyst" and Today's Date	"analyst" and Today's Date Displays	Result as expected	PASS	
4.3	14	Select Return Reason Code dropdown	"BROKEN-XSHIP Damaged-Replace Immediately"	List of Return Reason Code Displays	Result as expected	PASS	
5.8	15	Verify Selected Reason Code Displays and RMA details in the database updated accordingly		"BROKEN-XSHIP Damaged-Replace Immediately" Displays and RMA Details updated in the database	Result as expected	PASS	
	16	Select Credit, Replace, Repair? dropdown	"Replace"	List of Credit, Replace, Repair Displays	Result as expected	PASS	
5.8	17	Verify Selected Credit, Replace, Repair? Displays and RMA details in the database updated accordingly		"Replace" Displays and RMA Details update in the database	Result as expected	PASS	
	18	Select RMA Status Dropdown	"Dropped at Shop"	List of RMA Statuses Displays	Result as expected	PASS	
5.8	19	Verify Selected RMA Status Displays and RMA details in the database updated accordingly		"Dropped at Shop" Displays and RMA Details updated in the database	Result as expected	PASS	
5.2.1	20	Select Edit Button next to Additional Information/Special Instruction		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	21	Select "Confirm" on the Confirm Pop up		Additional Information/Special Instruction is editable and Button Text Changed to "Save"	Result as expected	PASS	
	22	Update Additional Information/Special Instruction Text	"Change was made Today"	"Change was made Today" Displays	Result as expected	PASS	
5.2.1.3	23	Select Save Button, select "Confirm" on the Confirm Popup, and verify that the change was saved in the database		"Change was made Today" Displays, RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	
	24	Select Product Dropdown	"Food, Grape"	List of Products Displays	Result as expected	PASS	
5.8	25	Verify Selected Product Displays and RMA details in the database updated accordingly		"Food, Grape" Displays and RMA Details update in the database	Result as expected	PASS	
5.2.1	26	Select Edit Button next to Return Label Tracking #		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	27	Select "Confirm" on the Confirm Pop up		Return Label Tracking # is editable and button Text Changed to "Save"	Result as expected	PASS	
	28	Update Return Label Tracking #	"#TestReturn005Changed"	"#TestReturn005Changed" Displays	Result as expected	PASS	
5.2.1.3	29	Select Save Button, select "Confirm" on the Confirm Popup, and verify that the change was saved in the database		"#TestReturn005Changed" Displays RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	
5.2.1	30	Select Edit Button next to Quantity		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	31	Select "Yes" on the Confirm Pop up		Quantity is editable and button Text Changed to "Save"	Result as expected	PASS	
4.7.3.1	32	Verify Quantity field only accept numerical character greater than 1 is entered	"-1, A"	Quatity field does not display non-numerical character and does not accept 0	Result as expected	PASS	
	33	Update Quantity	"1"	"1" Displays	Result as expected	PASS	
5.2.1.3	34	Select Save Button, select "Confirm" on the Confirm Popup, and verify that the change was saved in the database		"1" Displays RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	
5.2.1, 5.6.1	35	Select Edit Button next to Initial Evaluation		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	36	Select "Yes" on the Confirm Pop up		Initial Evaluation is editable and button Text Changed to "Save"	Result as expected	PASS	
	37	Update Initial Evaluation	"Test Initial Evaluation Worked Changed Today"	"Test Initial Evaluation Worked Changed Today" Displays	Result as expected	PASS	
5.2.1.3	38	Select Save Button, select "Confirm" on the Confirm Popup, and verify that the change was saved in the database		"Test Initial Evaluation Worked Changed Today" Displays RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	
5.6.2	39	Verify Edit button is not enabled next to Engineering Evaluation (Engineers and Admins Use Only)		Edit button is not selectable	Result as expected	PASS	
	40	Select Disposition Dropdown	"Scrap item, credit customer"	List of Disposition Displays	Result as expected	PASS	
5.8	41	Verify Selected Disposition Displays and RMA details in the database updated accordingly		"Scrap item, credit customer" Displays and RMA Details update in the database	Result as expected	PASS	
5.2.1, 5.6.1	42	Select Edit Button next to Disposition Notes		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	43	Select "Yes" on the Confirm Pop up		Disposition Notes is editable and button Text Changed to "Save"	Result as expected	PASS	
	44	Update Disposition Notes	"Test disposition Changed Today"	"Test disposition Changed Today" Displays	Result as expected	PASS	
5.2.1.3	45	Select Save Button, select "Confirm" on the Confirm Popup, and verify that the change was saved in the database		"Test disposition Changed Today" Displays RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	
5.2.1, 5.6.1	46	Select Edit Button next to Replacement/Repaired Product Tracking #		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	47	Select "Yes" on the Confirm Pop up		Replacement/Repaired Product Tracking # is editable and button Text Changed to "Save"	Result as expected	PASS	
	48	Update Replacement/Repaired Product Tracking #	"#TestReplaceRepair005Changed"	"#TestReplaceRepair005Changed" Displays	Result as expected	PASS	
5.2.1.3	49	Select Save Button, select "Confirm" on the Confirm Popup, and verify that the change was saved in the database		"#TestReplaceRepair005Changed" Displays RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	
	50	Select Replacement/Repaired Product Ship Date	The next day	Calendar Displays	Result as expected	PASS	
5.8	51	Verify Date changed to the next day and RMA details in the database updated accordingly		The next day date Displays and RMA Details update in the database	Result as expected	PASS	
	52	Uncheck Replacement/Repair Product Ship? Checkbox		Checkbox is set to unchecked	Result as expected	PASS	
5.1.1	53	Verify Progress Bar displays 60%		Progress Bar Displays 60%	Result as expected	PASS	

Title	Edit Buttons and Verify changes Saved
Description	The user Utilizes the Edit Button and makes changes. Form is saved with every change made. Progress bar is update depending on completion of each section.
Component Tested	RMA Detail Screen
Pre Condition	No Issue connecting to the Server and Database. Previously created RMA exists and user role is "admins".
Test ID	TC_16
Tester Name	Ryan McAllister-Grum

Tested Date	4/29/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
2.2.2	1	Select RMA Number from Table View	"0"	RMA Details Screen Displays	Result as expected	PASS	
5.0	2	Verify all values selected and entered that were previously saved matches the details screen	Saved RMA Details	All values that were stored into the database Displays correctly	Result as expected	PASS	
5.2.2	3	Verify RMA ID assigned under RMA # Label		Assigned RMA # Displays Correctly	Result as expected	PASS	
5.1, 5.1.1	4	Verify Progress Bar indicates correct percentage of completion.	Big Food Inc Previous Saved RMA	Progress Bar Displays 80%	Result as expected	PASS	
5.4, 5.4.1	5	Verify Details Area displays Customer information as default.		Customer's Information is Diplayed in the Details Area.	Result as expected	PASS	
5.2.4	6	Verify Age in Days is the difference between today's and created date	Created Date of RMA	Correct Age Displays	Result as expected	PASS	
5.2.5	7	Verify Created by and date	"analyst" and Created Date	"analyst" and Created Date Displays	Result as expected	PASS	
5.2.3	8	Verify Last Modified by and date	"analyst" and Creation Date	"analyst" and Date modified Displays	Result as expected	PASS	
5.0, 5.8	9	Verify all values selected and entered that were previously saved matches the details screen	Saved RMA Details	All values that were stored into the database Displays correctly	Result as expected	PASS	
4.6	10	Verify the current Owner "admin"		RMA Owner "admin" Displays	Result as expected	PASS	
	11	Select RMA Owner Dropdown		List of Employee name Displays	Result as expected	PASS	
5.2.6, 5.8	12	Verify Selected Owner Displays and RMA details in the database updated accordingly	"engineer"	"engineer" Displays in the field and RMA Details updated in the database	Result as expected	PASS	
5.2.3	13	Verify Last Modified by and date	"admin" and Today's Date	"admin" and Today's Date Displays	Result as expected	PASS	
4.3	14	Select Return Reason Code dropdown	"ERROR Customer ordered wrong item"	List of Return Reason Code Displays	Result as expected	PASS	
5.8	15	Verify Selected Reason Code Displays and RMA details in the database updated accordingly		"ERROR Customer ordered wrong item" Displays and RMA Details updated in the database	Result as expected	PASS	
	16	Select Credit, Replace, Repair? dropdown	"Credit"	List of Credit, Replace, Repair Displays	Result as expected	PASS	
5.8	17	Verify Selected Credit, Replace, Repair? Displays and RMA details in the database updated accordingly		"Credit" Displays and RMA Details update in the database	Result as expected	PASS	
5.2.7	18	Verify Replacement Details is collapse and values in the database under Replacement/Repaired Product Tracking #, Replacement/Repaired Product Ship Date and Replacement/Repair Product Ship? are all empty in the database		Replacement Details is collapse and disabled, values in the database under Replacement/Repaired Product Tracking #, Replacement/Repaired Product Ship Date and Replacement/Repair Product Ship? are all empty in the database	Result as expected	PASS	
5.1, 5.1.1	19	Verify Progress Bar indicates correct percentage of completion.	Big Food Inc Previous Saved RMA	Progress Bar Displays 75%	Result as expected	PASS	
	20	Select RMA Status Dropdown	"Waiting for Replacement"	List of RMA Statuses Displays	Result as expected	PASS	
5.8	21	Verify Selected RMA Status Displays and RMA details in the database updated accordingly		"Waiting for Replacement" Displays and RMA Details updated in the database	Result as expected	PASS	
5.2.1	22	Select Edit Button next to Additional Information/Special Instruction		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	23	Select "Yes" on the Confirm Pop up		Additional Information/Special Instruction is editable and Button Text Changed to "Save"	Result as expected	PASS	
	24	Update Additional Information/Special Instruction Text	"Change was made Today by Admin"	"Change was made Today by Admin" Displays	Result as expected	PASS	
5.2.1.3	25	Select Save Button, select "Confirm" on the Confirm Popup, and verify that the change was saved in the database		"Change was made Today by Admin" Displays, RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	
	26	Select Product Dropdown	"Food, Grape"	List of Products Displays	Result as expected	PASS	
5.8	27	Verify Selected Product Displays and RMA details in the database updated accordingly		"Food, Grape" Displays and RMA Details update in the database	Result as expected	PASS	
5.2.1	28	Select Edit Button next to Return Label Tracking #		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	29	Select "Yes" on the Confirm Pop up		Return Label Tracking # is editable and button Text Changed to "Save"	Result as expected	PASS	
	30	Update Return Label Tracking #	"#TestReturn005ChangedAdmin"	"#TestReturn005Changed" Displays	Result as expected	PASS	
5.2.1.3	31	Select Save Button, select "Confirm" on the Confirm Popup, and verify that the change was saved in the database		"#TestReturn005ChangedAdmin" Displays RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	
5.2.1	32	Select Edit Button next to Quantity		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	33	Select "Yes" on the Confirm Pop up		Quantity is editable and button Text Changed to "Save"	Result as expected	PASS	
4.7.3.1	34	Verify Quantity field only accept numerical character greater than 1 is entered	"-1, A"	Quatity field does not display non-numerical character and does not accept 0	Result as expected	PASS	
	35	Update Quantity	"1"	"1" Displays	Result as expected	PASS	
5.2.1.3	36	Select Save Button, select "Confirm" on the Confirm Popup, and verify that the change was saved in the database		"1" Displays RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	
5.2.1, 5.6.1	37	Select Edit Button next to Initial Evaluation		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	38	Select "Yes" on the Confirm Pop up		Initial Evaluation is editable and button Text Changed to "Save"	Result as expected	PASS	
	39	Update Initial Evaluation	"Test Initial Evaluation Worked Changed Today by Admin"	"Test Initial Evaluation Worked Changed Today by Admin" Displays	Result as expected	PASS	
5.2.1.3	40	Select Save Button, select "Confirm" on the Confirm Popup, and verify that the change was saved in the database		"Test Initial Evaluation Worked Changed Today" Displays RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	
5.2.1, 5.6.1	41	Select Edit Button next to Engineering Evaluation		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	42	Select "Yes" on the Confirm Pop up		Engineering Evaluation is editable and button Text Changed to "Save"	Result as expected	PASS	
	43	Update Engineering Evaluation	"Test Engineering Evaluation Worked Changed Today by Admin"	"Test Engineering Evaluation Worked Changed Today by Admin" Displays	Result as expected	PASS	
5.2.1.3	44	Select Save Button, select "Confirm" on the Confirm Popup, and verify that the change was saved in the database		"Test Engineering Evaluation Worked Changed Today by Admin" Displays RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	
5.1.1	45	Verify Progress Bar displays 100%		Progress Bar Displays 100%	Result as expected	PASS	
	46	Select Disposition Dropdown	"Put item back into inventory, credit customer"	List of Disposition Displays	Result as expected	PASS	
5.8	47	Verify Selected Disposition Displays and RMA details in the database updated accordingly		"Put item back into inventory, credit customer" Displays and RMA Details update in the database	Result as expected	PASS	
5.2.1, 5.6.1	48	Select Edit Button next to Disposition Notes		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	49	Select "Yes" on the Confirm Pop up		Disposition Notes is editable and button Text Changed to "Save"	Result as expected	PASS	
	50	Update Disposition Notes	"Test disposition Changed Today by Admin"	"Test disposition Changed Today by Admin" Displays	Result as expected	PASS	
5.2.1.3	51	Select Save Button, select "Confirm" on the Confirm Popup, and verify that the change was saved in the database		"Test disposition Changed Today by Admin" Displays RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	

Title	**Engineer User only able to Edit Engineering Evaluation**
Description	The user utilizes the Edit Button and makes changes. Form is saved with every change made. Progress bar is update depending on completion of each section.
Component Tested	RMA Detail Screen
Pre Condition	No Issue connecting to the Server and Database. Previously created RMA exists and user role is "engineers".
Test ID	TC_17
Tester Name	Ryan McAllister-Grum
Tested Date	4/29/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
2.2.2	1	Select RMA Number from Table View	"0"	RMA Details Screen Displays	Result as expected	PASS	
5.0	2	Verify all values selected and entered that were previously saved matches the details screen	Saved RMA Details	All values that were stored into the database Displays correctly	Result as expected	PASS	
5.2.2	3	Verify RMA ID assigned under RMA # Label		Assigned RMA # Displays Correctly	Result as expected	PASS	
5.1, 5.1.1	4	Verify Progress Bar indicates correct percentage of completion.	Big Food Inc Previous Saved RMA	Progress Bar Displays 100%	Result as expected	PASS	
5.4, 5.4.1	5	Verify Details Area displays Customer information as default.		Customer's Information is Diplayed in the Details Area.	Result as expected	PASS	
5.2.4	6	Verify Age in Days is the difference between today's and created date	Created Date of RMA	Correct Age Displays	Result as expected	PASS	

5.2.5	7	Verify Created by and date	"Analyst" and Created Date	"Analyst" and Created Date Displays	Result as expected	PASS	
5.2.3	8	Verify Last Modified by and date	"Admin" and Correct Date	"Admin" and date modified Displays	Result as expected	PASS	
5.0, 5.8	9	Verify all values selected and entered that were previously saved matches the details screen	Saved RMA Details	All values that were stored into the database Displays correctly	Result as expected	PASS	
5.6.2	10	Verify No other Edit buttons or Dropdowns are selectable besides Edit button next to Engineering Evaluation		All other fields, buttons and dropdowns are not selectable	Result as expected	PASS	
5.2.1, 5.6.3	11	Select Edit Button next to Engineering Evaluation		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	12	Select "Yes" on the Confirm Pop up		Engineering Evaluation is editable and button Text Changed to "Save"	Result as expected	PASS	
	13	Delete all text in the Engineering Evaluation field	Empty	Nothing Displays	Result as expected	PASS	
5.2.1.3	14	Select Save Button, select "Confirm" on the Confirm Popup, and verify that the change was saved in the database		Empty value Displays RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	
5.1.1	15	Verify Progress Bar displays 75%		Progress Bar Displays 75%	Result as expected	PASS	

Title	**Home Button**
Description	The user selects the Home Button and is returned to the RMA List View.
Component Tested	RMA Detail Screen
Pre Condition	No Issue connecting to the Server and Database. Previously created RMA exists and user role is "analysts".
Test ID	TC_18
Tester Name	Ryan McAllister-Grum
Tested Date	4/29/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
2.2.2	1	Select RMA Number from Table View	"0"	RMA Details Screen Displays	Result as expected	PASS	
5.0	2	Verify all values selected and entered that were previously saved matches the details screen	Saved RMA Details	All values that were stored into the database Displays correctly	Result as expected	PASS	
5.2.2	3	Verify RMA ID assigned under RMA # Label		Assigned RMA # Displays Correctly	Result as expected	PASS	
5.1, 5.1.1	4	Verify Progress Bar indicates correct percentage of completion.	Big Food Inc Previous Saved RMA	Progress Bar Displays 75%	Result as expected	PASS	
5.4, 5.4.1	5	Verify Details Area displays Customer information as default.		Customer's Information is Diplayed in the Details Area.	Result as expected	PASS	
5.2.4	6	Verify Age in Days is the difference between today's and created date	Created Date of RMA	Correct Age Displays	Result as expected	PASS	
5.2.5	7	Verify Created by and date	"Analyst" and Created Date	"Analyst" and Created Date Displays	Result as expected	PASS	
5.2.3	8	Verify Last Modified by and date	"engineer" and Correct Date	"Admin" and date modified Displays	Result as expected	PASS	
5.0, 5.8	9	Verify all values selected and entered that were previously saved matches the details screen	Saved RMA Details	All values that were stored into the database Displays correctly	Result as expected	PASS	
5.6.2	10	Select the Home Button and select "Confirm" on the Confirm Popup		RMA List View Screen Displays	Result as expected	PASS	

RMA List View Test

Title	Button functionality and Screen Display for Analyst
Description	The user test all component functionality and displays requirements.
Component Tested	Home Screen, RMA Detail Screen and New Form
Pre Condition	No Issue connecting to the Server and Database and the user has access. Table displays previously created RMAs. User logged in is "analyst"
Tested ID	TC_19
Tester Name	Ryan McAllister-Grum
Tested Date	4/29/2021
Execusion Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
2.0	1	Login to the application		Home Screen Displays	Result as expected	PASS	
2.1	2	Verify total items in the list table and the last updated date and time		Number of total item in the list table and last updated date and time displays	Result as expected	PASS	
2.2, 2.2.1	3	Verify Table of all RMAs display	Created RMA	Table of all RMA Displays in respective columns	Result as expected	PASS	
2.2.2	4	Double Click on RMA Row		RMA Details Screen for Corresponding RMA Displays	Result as expected	PASS	
5.6.3		Verify Engineering Evaluation Edit Button is disabled		Engineering Evaluation Edit Button is disabled	Result as expected	PASS	
5.7	5	From RMA Details Screen Select Home Button to return to the Home Screen		Home Screen Displays	Result as expected	PASS	
2.3	6	Search for an RMA by entering an RMA ID #	"0"	Entered RMA ID Displays	Result as expected	PASS	
2.3.1	7	Select Enter button on the Keyboard		Table is updated and RMA ID searched displays	Result as expected	PASS	
2.4, 2.4.1	8	Select New RMA Button		New Empty RMA Form Displays	Result as expected	PASS	
4.17, 4.10.3	9	Select Close Button		Confirmation pop up displays	Result as expected	PASS	
	10	Yes button Selected	Yes	Confirmation pop up closes and user is returned to the home screen.	Result as expected	PASS	

Title	Button functionality and Screen Display for Admin
Description	The user test all component functionality and displays requirements.
Component Tested	Home Screen, RMA Details and New Form
Pre Condition	No Issue connecting to the Server and Database and the user has access. Table displays previously created RMAs. User logged in is "admin"
Tested ID	TC_20
Tester Name	Ryan McAllister-Grum
Tested Date	4/29/2021
Execusion Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
2.0	1	Login to the application		Home Screen Displays	Result as expected	PASS	
2.1	2	Verify total items in the list table and the last updated date and time		Number of total item in the list table and last updated date and time displays	Result as expected	PASS	
2.2, 2.2.1	3	Verify Table of all RMAs display	Created RMA	Table of all RMA Displays in respective columns	Result as expected	PASS	
2.2.2	4	Double Click on RMA Row		RMA Details Screen for Corresponding RMA Displays	Result as expected	PASS	
5.6.3		Verify Initial Evaluation and Engineering Evaluation Edit Buttons are enabled		Initial Evaluation and Engineering Evaluation Edit Buttons are enabled	Result as expected	PASS	
5.7	5	From RMA Details Screen Select Home Button to return to the Home Screed		Home Screen Displays	Result as expected	PASS	
2.3	6	Search for an RMA by entering an RMA ID #	"0"	Entered RMA ID Displays	Result as expected	PASS	
2.3.1	7	Select Enter button on the Keyboard		Table is updated and RMA ID searched displays	Result as expected	PASS	
2.4, 2.4.1	8	Select New RMA Button		New Empty RMA Form Displays	Result as expected	PASS	
4.17, 4.10.3	9	Select Close Button		Confirmation pop up displays	Result as expected	PASS	
	10	Yes button Selected	Yes	Confirmation pop up closes and user is returned to the home screen.	Result as expected	PASS	

Title	Button functionality and Screen Display for Engineer
Description	The user test all component functionality and displays requirements.
Component Tested	Home Screen
Pre Condition	No Issue connecting to the Server and Database and the user has access. Table displays previously created RMAs. User logged in is "engineer"
Tested ID	TC_21
Tester Name	Ryan McAllister-Grum
Tested Date	4/29/2021
Execusion Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
2.0	1	Login to the application		Home Screen Displays	Result as expected	PASS	
2.1	2	Verify total items in the list table and the last updated date and time		Number of total item in the list table and last updated date and time displays	Result as expected	PASS	
2.2, 2.2.1	3	Verify Table of all RMAs display	Created RMA	Table of all RMA Displays in respective columns	Result as expected	PASS	
2.2.2	4	Double Click on RMA Row		RMA Details Screen for Corresponding RMA Displays	Result as expected	PASS	
5.6.3		Verify all Buttons and Dropdowns besides Engineering Evaluation Edit Button are disabled		Only Edit Button next to Engineering Evaluation Field is enabled	Result as expected	PASS	
5.7	5	From RMA Details Screen Select Home Button to return to the Home Screed		Home Screen Displays	Result as expected	PASS	
2.3, 2.3.1	6	Search for an RMA by entering an RMA ID #	"0"	Entered RMA ID Displays	Result as expected	PASS	
2.4.2	8	Verify New RMA button is disabled		New RMA Button is disabled	Result as expected	PASS	
2.7.1	9	Verify Delete Button is disabled		Delete Button is disabled	Result as expected	PASS	

Title	Create a New RMA and Verify List Updates
Description	The user creates a new RMA and then deletes it.
Component Tested	Home Screen and Create New RMA form
Pre Condition	No Issue connecting to the Server and Database and the user has access. User logged in is "analyst"
Tested ID	TC_22
Tester Name	Ryan McAllister-Grum
Tested Date	4/29/2021
Execusion Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
2.0	1	Login to the application		Home Screen Displays	Result as expected	PASS	
2.1	2	Verify total items in the list table and the last updated date and time		Number of total item in the list table and last updated date and time displays	Result as expected	PASS	
2.2, 2.2.1	3	Verify Table of all RMAs display	Created RMA	Table of all RMA Displays in respective columns	Result as expected	PASS	
2.4, 2.4.1	4	Select New RMA Button		New Empty RMA Form Displays	Result as expected	PASS	
	5	Complete all required Fields		All Required Fields are complete	Result as expected	PASS	
4.10, 4.10.1	6	Select Save & Close Button		Confirmation pop up displays	Result as expected	PASS	
4.10.1	7	Confirm Button Selected		Confirmation pop up closes and New RMA is created and stored in the database. User is returned to the Home Screen.	Result as expected	PASS	
2.5.1	8	Verify created RMA displays in the Table		Created RMA displays in the Table	Result as expected	PASS	
2.1	9	Verify total items in the list table and the last updated date and time updated correctly		Number of total item in the list table and last updated date and time displays correctly	Result as expected	PASS	
2.7.2	10	Select the Delete Button		No change occurs	Result as expected	PASS	
2.7, 2.7.2	11	Check the Box in the "Delete?" Column in the Table for the recently created RMA		Box is marked with a check displays	Result as expected	PASS	
	12	Select Delete Button		Delete Confirmation Pop Up displays	Result as expected	PASS	
2.7.3	13	Select Cancel button		Delete Confirmation hides and User is returned to Home Screen no change occurred	Result as expected	PASS	

	14	Select Delete Button		Delete Confirmation Pop Up displays	Result as expected	PASS	
2.7.3	15	Select Confirm button		Delete Confirmation hides and User is returned to Home Screen	Result as expected	PASS	
	16	Verify newly Created RMA is Deleted and removed from the Database		Newly created RMA does not display in the Table and removed from the database	Result as expected	PASS	
2.1	17	Verify total items in the list table and the last updated date and time updated correctly		Number of total item in the list table and last updated date and time displays correctly	Result as expected	PASS	

Title	**Delete Multiple RMAs**
Description	The user selectes multiple RMAs and deletes it.
Component Tested	Home Screen
Pre Condition	No Issue connecting to the Server and Database and the user has access. User logged in is "analyst"
Tested ID	TC_23
Tester Name	Ryan McAllister-Grum
Tested Date	4/29/2021
Execusion Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
2.0	1	Login to the application		Home Screen Displays	Result as expected	PASS	
2.1	2	Verify total items in the list table and the last updated date and time		Number of total item in the list table and last updated date and time displays	Result as expected	PASS	
2.2, 2.2.1	3	Verify Table of all RMAs display	Created RMA	Table of all RMA Displays in respective columns	Result as expected	PASS	
2.7, 2.7.2	4	Check the Box in the Delete Column of the Table of multiple RMAs	2 RMAs checked	Box is marked with a check displays	Result as expected	PASS	
	5	Select Delete Button		Delete Confirmation Pop Up displays	Result as expected	PASS	
2.7.3	7	Select Cancel button		Delete Confirmation hides and User is returned to Home Screen no change occurred	Result as expected	PASS	
	8	Select Delete Button		Delete Confirmation Pop Up displays	Result as expected	PASS	
2.7.3	9	Select Confirm button		Delete Confirmation hides and User is returned to Home Screen	Result as expected	PASS	
	10	Verify Selected RMAs are deleted and removed from the Database		Selected RMAs does not display in the Table and removed from the database	Result as expected	PASS	
2.1	11	Verify total items in the list table and the last updated date and time updated correctly		Number of total item in the list table and last updated date and time displays correctly	Result as expected	PASS	

Title	**Update RMA and Attempt to Delete as Engineer**
Description	The user updates RMA and attempt to delete it.
Component Tested	Home Screen and RMA Details Screen
Pre Condition	No Issue connecting to the Server and Database and the user has access. User logged in is "engineer"
Tested ID	TC_24
Tester Name	Ryan McAllister-Grum
Tested Date	4/29/2021
Execusion Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
2.0	1	Login to the application		Home Screen Displays	Result as expected	PASS	
2.1	2	Verify total items in the list table and the last updated date and time		Number of total item in the list table and last updated date and time displays	Result as expected	PASS	
2.2.2	3	Double Click on RMA Row		RMA Details Screen for Corresponding RMA Displays	Result as expected	PASS	
5.6.3	4	Verify Engineering Evaluation Edit Button is the only enabled button		Engineering Evaluation Edit Button is the only enabled button	Result as expected	PASS	
	5	Updates the Engineering Evaluation field and saves changes	"Engineering changed today"	"Engineering changed today" Displays in the Engineering Evaluation field	Result as expected	PASS	
5.7	6	From RMA Details Screen Select Home Button to return to the Home Screen		Home Screen Displays	Result as expected	PASS	
	7	Verify changes made to RMA in the Table		Changes in the RMA Displays	Result as expected	PASS	
2.7, 2.7.2	8	Check the Box in the Delete Column of the recently change RMA	Check Box	Box is marked with a check displays	Result as expected	PASS	
2.7.3	9	Verify Delete Button is not enabled		Delete Button is not enabled	Result as expected	PASS	

Title	**Update RMA Then Delete as Analyst**
Description	The User updates the RMA then deletes it and then exits the application.
Component Tested	Home Screen and RMA Details Screen
Pre Condition	No Issue connecting to the Server and Database and the user has access. User logged in is "analyst"
Tested ID	TC_25
Tester Name	Ryan McAllister-Grum
Tested Date	4/29/2021
Execusion Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
2.0	1	Login to the application		Home Screen Displays	Result as expected	PASS	
2.1	2	Verify total items in the list table and the last updated date and time		Number of total item in the list table and last updated date and time displays	Result as expected	PASS	
2.2, 2.2.1	3	Verify Table of all RMAs display	Created RMA	Table of all RMA Displays in respective columns	Result as expected	PASS	
2.2.2	4	Double Click on RMA Row		RMA Details Screen for Corresponding RMA Displays	Result as expected	PASS	
5.6.3	5	Verify Engineering Evaluation Edit Button is disabled		Engineering Evaluation Edit Button is disabled	Result as expected	PASS	
	6	Update the RMA Status	Change to any other RMA Status	New RMA Status Displays	Result as expected	PASS	
5.7	7	From RMA Details Screen Select Home Button to return to the Home Screen		Home Screen Displays	Result as expected	PASS	
	8	Verify changes made to RMA in the Table		RMA Status Change Displays	Result as expected	PASS	
2.7, 2.7.2	9	Check the Box in the Delete Column of the recently change RMA	Check Box	Box is marked with a check displays	Result as expected	PASS	
	10	Select Delete Button		Delete Confirmation Pop Up displays	Result as expected	PASS	
2.7.3	11	Select Cancel button		Delete Confirmation hides and User is returned to Home Screen no change occurred	Result as expected	PASS	
	12	Select Delete Button		Delete Confirmation Pop Up displays	Result as expected	PASS	
2.7.3	13	Select Confirm button		Delete Confirmation hides and User is returned to Home Screen	Result as expected	PASS	
	14	Verify Selected RMAs are deleted and removed from the Database		Selected RMAs does not display in the Table and removed from the database	Result as expected	PASS	
2.1	15	Verify total items in the list table and the last updated date and time updated correctly		Number of total item in the list table and last updated date and time displays correctly	Result as expected	PASS	
2.6	16	Select the Exit Button		Exit Confirm Pop Up displays	Result as expected	PASS	
2.6.1	17	Select Cancel button		Exit Confirmation hides and User is returned to Home Screen	Result as expected	PASS	
	18	Select Exit Button		Exit Confirmation Pop Up displays	Result as expected	PASS	
2.6.1	19	Select Confirm button		Delete Confirmation hides and Application is closed	Result as expected	PASS	

End to End Test

Component Tested	End to End test
Description	Test from login to Close with all username involved.
Component Tested	Login Screen, Home Screen, RMA Details Screen, New Form
Pre Condition	No Issue connecting to the Server and Database and the user has access, and previous created RMA exist.
Test ID	TC_26
Tester Name	Ryan McAllister-Grum
Tested Date	4/29/2021
Execution Result	PASS

Requirement	Steps	Execution Steps	Test Data	Expected Result	Actual Results	Pass/Fail	Comments
	1	Launch the Application		Login Screen Displays	Result as expected	PASS	
1.1	2	Enter Username	"analyst"	Username is correct	Result as expected	PASS	
1.2	3	Enter Password	Password Valid	Password is correct	Result as expected	PASS	
1.4	4	Select Server Instance	Instance Valid	Valid instance displays the Dropdown List and Connect button is enabled	Result as expected	PASS	
1.5, 2.0	5	Select Connect Button		Access granted and Home Screen Displays	Result as expected	PASS	
2.4, 2.4.1	6	Select New RMA Button		New RMA Form Displays	Result as expected	PASS	
4.18	7	Verify Business Name, PO Number, and Product dropdowns are not enabled.		Dropdowns are not Enabled	Result as expected	PASS	
4.1	8	Select Customer Name dropdown	"Big Food Inc"	List of Customer Name Displays	Result as expected	PASS	
4.18.1	9	Verify Selected Customer Name displays		"Big Food Inc" Displays and Business Name Enabled	Result as expected	PASS	
4.2	10	Select Business Name dropdown	"George's Corner Store"	List of Business Name Associated with Customer Displays Only	Result as expected	PASS	
4.18.2	11	Verify Selected Business Name displays		"George's Corner Store" Displays and PO Number Enabled	Result as expected	PASS	
4.12, 4.12.1	12	Verify Shipping Address auto populated	"Big Food Inc. George's Corner Store 63 S Mission Rd Eastborough Sedgwick KS 67207 United States of America Phone: 621-187-3888"	"Big Food Inc. George's Corner Store 63 S Mission Rd Eastborough Sedgwick KS 67207 United States of America Phone: 621-187-3888" Displays	Result as expected	PASS	
4.11	13	Select P.O. Number dropdown	"3C2218QN"	List of P.O. Numbers Associated with Business Displays Only	Result as expected	PASS	
4.18.3	14	Verify Selected P.O. Number displays		"3C2218QN" Displays and Product Enabled	Result as expected	PASS	
4.3	15	Select Return Reason Code dropdown	"BROKEN-RET Damaged-Replace when returned"	List of Return Reason Code Displays	Result as expected	PASS	
	16	Verify Selected Reason Code displays		"BROKEN-RET Damaged-Replace when returned" Displays	Result as expected	PASS	
4.4	17	Select Credit, Replace, Repair? dropdown	"Repair"	List of Credit, Replace, Repair Displays	Result as expected	PASS	
	18	Verify Selected Credit, Replace, Repair? displays		"Repair" Displays	Result as expected	PASS	
4.5, 4.5.1	19	Select RMA Status Dropdown and "Close" does not Display	"Diagnose"	List of RMA Statuses Displays besides "Closed"	Result as expected	PASS	
	20	Verify Selected RMA Status Displays		"Diagnose" Displays	Result as expected	PASS	
4.13	21	Enter text in the the Additional Information/Special Instruction	"This is a test, it is only a test"	Entered Text Displays "This is a test, it is only a test"	Result as expected	PASS	
4.7, 4.7.1	22	Select Product dropdown	"Food, Apple"	List of Products Displays	Result as expected	PASS	
	23	Verify Selected Product displays		"Food, Apple" Displays	Result as expected	PASS	
4.7.2	24	Enter Quantity	"5"	Entered Quantity Displays	Result as expected	PASS	
4.7.3	25	Enter Return Label Tracking number	"#TestReturn005"	Entered Return Label Tracking # Displays	Result as expected	PASS	
4.8	26	Enter Initial Evaluation	"Test Initial Evaluation Worked"	Entered Initial Evaluation Displays	Result as expected	PASS	
4.9.1	27	Select Disposition dropdown	"Repair and add to inventory, credit customer"	List of Disposition Displays	Result as expected	PASS	
	28	Verify Selected Disposition displays		"Repair and add to inventory, credit customer" Displays	Result as expected	PASS	
4.9.2	29	Enter Disposition Notes	"Test disposition"	Entered Disposition Displays	Result as expected	PASS	
4.14	30	Enter Replacement/Repair Product Tracking #	"#TestReplaceRepair005"	Entered Tracking # Displays	Result as expected	PASS	
4.15	31	Select a date from the calendar for Replacement/Repair Product Ship Date	Select today's date	Today's Date displays	Result as expected	PASS	
4.16	32	Check the Replacement/Repair Product Ship? Box		Checkbox is filled	Result as expected	PASS	
4.10, 4.10.1	33	Select Save Button		Confirmation pop up displays	Result as expected	PASS	
4.10.1	34	Confirm Button Selected		Confirmation pop up closes and New RMA is created and stored in the database. Focus is returned to NEW RMA form screen.	Result as expected	PASS	
3.0, 5.2.2	35	Verify in the Database New RMA Save Successful	Completed RMA Form	New RMA create and RMAID auto generated and saved in the database	Result as expected	PASS	
3.8,3.9	36	Verify all values selected and entered were saved correctly into the database	All RMA entered values	All values were stored into the database	Result as expected	PASS	
3.8,3.9	37	Verify RMA ID auto generated and assigned	Auto Generated unique ID	Unique RMA ID auto generated and stored	Result as expected	PASS	
5.2.2	38	Verify Owner auto assigned to Creator	Auto assigned to Owner	Owner and Creator are stored in the database as the same	Result as expected	PASS	
4.10, 4.10.3	39	Select Cancel Button		Confirmation pop up displays	Result as expected	PASS	
	40	Confirm button Selected	Confirm	Confirmation pop up closes and user is returned to the home screen.	Result as expected	PASS	
2.1	41	Verify total items in the list table and the last updated date and time		Number of total item in the list table and last updated date and time displays	Result as expected	PASS	
	42			Table View is refreshed	Result as expected	PASS	
2.5.1	43	Verify create RMA displays in the Table		Create RMA displays in the Table	Result as expected	PASS	
2.1	44	Verify total items in the list table and the last updated date and time updated correctly		Number of total item in the list table and last updated date and time displays correctly	Result as expected	PASS	
2.2.2	45	Double Click on RMA Row		RMA Details Screen for Corresponding RMA Displays	Result as expected	PASS	
5.0	46	Verify all values selected and entered that were previously saved matches the details screen	Saved RMA Details	All values that were stored into the database Displays correctly	Result as expected	PASS	
5.2.2	47	Verify RMA ID assigned under RMA # Label		Assigned RMA # Displays Correctly	Result as expected	PASS	
5.1, 5.1.1	48	Verify Progress Bar indicates correct percentage of completion.	Big Food Inc Previous Saved RMA	Progress Bar Displays 80%	Result as expected	PASS	
5.4, 5.4.1	49	Verify Details Area displays Customer information as default.		Customer's Information is Diplayed in the Details Area.	Result as expected	PASS	
5.2.4	50	Verify Age in Days is the difference between today's and created date	Created Date of RMA	Correct Age Displays	Result as expected	PASS	
5.2.5	51	Verify Created by and date	"analyst" and Created Date	"analyst" and Created Date Displays	Result as expected	PASS	
5.2.3	52	Verify Last Modified by and date	"analyst" and Creation Date	"analyst" and Same Date as Created Date Displays	Result as expected	PASS	
5.0, 5.8	53	Verify all values selected and entered that were previously saved matches the details screen	Saved RMA Details	All values that were stored into the database Displays correctly	Result as expected	PASS	
4.6	54	Verify the Default Owner is set to the Creator of the RMA	"analyst"	RMA Owner "analyst" Displays	Result as expected	PASS	
	55	Select RMA Owner Dropdown		List of Employee name Displays	Result as expected	PASS	
5.2.6, 5.8	56	Verify Selected Owner Displays and RMA details in the database updated accordingly	"admin"	"admin" Displays in the field and RMA Details updated in the database	Result as expected	PASS	
5.2.3	57	Verify Last Modified by and date	"analyst" and Today's Date	"analyst" and Today's Date Displays	Result as expected	PASS	
4.3	58	Select Return Reason Code dropdown	"BROKEN-XSHIP Damaged-Replace Immediately"	List of Return Reason Code Displays	Result as expected	PASS	
5.8	59	Verify Selected Reason Code Displays and RMA details in the database updated accordingly		"BROKEN-XSHIP Damaged-Replace Immediately" Displays and RMA Details updated in the database	Result as expected	PASS	
	60	Select Credit, Replace, Repair? dropdown	"Replace"	List of Credit, Replace, Repair Displays	Result as expected	PASS	
5.8	61	Verify Selected Credit, Replace, Repair? Displays and RMA details in the database updated accordingly		"Replace" Displays and RMA Details update in the database	Result as expected	PASS	
	62	Select RMA Status Dropdown	"Dropped at Shop"	List of RMA Statuses Displays	Result as expected	PASS	
5.8	63	Verify Selected RMA Status Displays and RMA details in the database updated accordingly		"Dropped at Shop" Displays and RMA Details updated in the database	Result as expected	PASS	

5.2.1	64	Select Edit Button next to Additional Information/Special Instruction		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	65	Select "Yes" on the Confirm Pop up		Additional Information/Special Instruction is editable and Button Text Changed to "Save"	Result as expected	PASS	
	66	Update Additional Information/Special Instruction Text	"Change was made Today"	"Change was made Today" Displays	Result as expected	PASS	
5.2.1.3	67	Select Save Button and verify that the change was saved in the database		"Change was made Today" Displays, RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	
	68	Select Product Dropdown	"Food, Grape"	List of Products Displays	Result as expected	PASS	
5.8	69	Verify Selected Product Displays and RMA details in the database updated accordingly		"Food, Grape" Displays and RMA Details update in the database	Result as expected	PASS	
5.2.1	70	Select Edit Button next to Return Label Tracking #		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	71	Select "Yes" on the Confirm Pop up		Return Label Tracking # is editable and button Text Changed to "Save"	Result as expected	PASS	
	72	Update Return Label Tracking #	"#TestReturn005Changed"	"#TestReturn005Changed" Displays	Result as expected	PASS	
5.2.1.3	73	Select Save Button and verify that the change was saved in the database		"#TestReturn005Changed" Displays RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	
5.2.1	74	Select Edit Button next to Quantity		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	75	Select "Yes" on the Confirm Pop up		Quantity is editable and button Text Changed to "Save"	Result as expected	PASS	
4.7.3.1	76	Verify Quantity field only accept numerical character greater than 1 is entered	"-1, A"	Quatity field does not display non-numerical character and does not accept 0	Result as expected	PASS	
	77	Update Quantity	"1"	"1" Displays	Result as expected	PASS	
5.2.1.3	78	Select Save Button and verify that the change was saved in the database		"1" Displays RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	
5.2.1, 5.6.1	79	Select Edit Button next to Initial Evaluation		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	80	Select "Yes" on the Confirm Pop up		Initial Evaluation is editable and button Text Changed to "Save"	Result as expected	PASS	
	81	Update Initial Evaluation	"Test Initial Evaluation Worked Changed Today"	"Test Initial Evaluation Worked Changed Today" Displays	Result as expected	PASS	
5.2.1.3	82	Select Save Button and verify that the change was saved in the database		"Test Initial Evaluation Worked Changed Today" Displays RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	
5.6.2	83	Verify Edit button is not enabled next to Engineering Evaluation (engineers and admins Use Only)		Edit button is not selectable	Result as expected	PASS	
	84	Select Disposition Dropdown	"Scrap item, credit customer"	List of Disposition Displays	Result as expected	PASS	
5.8	85	Verify Selected Disposition Displays and RMA details in the database updated accordingly		"Scrap item, credit customer" Displays and RMA Details update in the database	Result as expected	PASS	
5.2.1, 5.6.1	86	Select Edit Button next to Disposition Notes		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	87	Select "Confirm" on the Confirm Pop up		Disposition Notes is editable and button Text Changed to "Save"	Result as expected	PASS	
	88	Update Disposition Notes	"Test disposition Changed Today"	"Test disposition Changed Today" Displays	Result as expected	PASS	
5.2.1.3	89	Select Save Button and verify that the change was saved in the database		"Test disposition Changed Today" Displays RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	
5.2.1, 5.6.1	90	Select Edit Button next to Replacement/Repaired Product Tracking #		Confirmation Pop Up Displays	Result as expected	PASS	
5.2.1.1	91	Select "Yes" on the Confirm Pop up		Replacement/Repaired Product Tracking # is editable and button Text Changed to "Save"	Result as expected	PASS	
	92	Update Replacement/Repaired Product Tracking #	"#TestReplaceRepair005Changed"	"#TestReplaceRepair005Changed" Displays	Result as expected	PASS	
5.2.1.3	93	Select Save Button and verify that the change was saved in the database		"#TestReplaceRepair005Changed" Displays RMA Details updated in the database, and Button Text Changed to "Edit"	Result as expected	PASS	
	94	Select Replacement/Repaired Product Ship Date	The next day	Calendar Displays	Result as expected	PASS	
5.8	95	Verify Date changed to the next day and RMA details in the database updated accordingly		The next day date Displays and RMA Details update in the database	Result as expected	PASS	
	96	Uncheck Replacement/Repair Product Ship? Checkbox		Checkbox is set to unchecked	Result as expected	PASS	
5.1.1	97	Verify Progress Bar displays 60%		Progress Bar Displays 60%	Result as expected	PASS	
2.6	98	Select the Exit Button		Exit Confirm Pop Up displays	Result as expected	PASS	
2.6.1	99	Select No button		Exit Confirmation hides and User is returned to Home Screen	Result as expected	PASS	
	100	Select Exit Button		Exit Confirmation Pop Up displays	Result as expected	PASS	
2.6.1	101	Select Confirm button		Delete Confirmation hides and Application is closed	Result as expected	PASS	
	102	Launch the Application		Login Screen Displays	Result as expected	PASS	
1.1	103	Enter Username	"engineer"	Username is correct	Result as expected	PASS	
1.2	104	Enter Password	Password Valid	Password is correct	Result as expected	PASS	
1.4	105	Select Server Instance	Instance Valid	Valid instance displays the Dropdown List and Connect button is enabled	Result as expected	PASS	
1.5, 2.0	106	Select Connect Button		Access granted and Home Screen Displays	Result as expected	PASS	
2.2.2	107	Double Click on RMA Row that matched RMA Id # from previous Entry	"0"	RMA Details Screen for Corresponding RMA Displays	Result as expected	PASS	
5.6.3	108	Verify Engineering Evaluation Edit Button is the only enabled button		Engineering Evaluation Edit Button is the only enabled button	Result as expected	PASS	
	109	Updates the Engineering Evaluation field and saves changes	"Engineering changed today"	"Engineering changed today" Displays in the Engineering Evaluation field	Result as expected	PASS	
5.1.1	110	Verify Progress Bar displays 80%		Progress Bar Displays 80%	Result as expected	PASS	
5.7	111	From RMA Details Screen Select Home Button to return to the Home Screen		Home Screen Displays	Result as expected	PASS	
	112	Verify changes made to RMA in the Table		Changes in the RMA Displays	Result as expected	PASS	
2.6	113	Select the Exit Button		Exit Confirm Pop Up displays	Result as expected	PASS	
2.6.1	114	Select No button		Exit Confirmation hides and User is returned to Home Screen	Result as expected	PASS	
	115	Select Exit Button		Exit Confirmation Pop Up displays	Result as expected	PASS	
2.6.1	116	Select Confirm button		Delete Confirmation hides and Application is closed	Result as expected	PASS	
	117	Launch the Application		Login Screen Displays	Result as expected	PASS	
1.1	118	Enter admin	"admin"	Username is correct	Result as expected	PASS	
1.2	119	Enter Password	Password Valid	Password is correct	Result as expected	PASS	
1.4	120	Select Server Instance	Instance Valid	Valid instance displays the Dropdown List and Connect button is enabled	Result as expected	PASS	
1.5, 2.0	121	Select Connect Button		Access granted and Home Screen Displays	Result as expected	PASS	
2.2.2	122	Double Click on RMA Row that matched RMA Id # from previous Entry	"0"	RMA Details Screen for Corresponding RMA Displays	Result as expected	PASS	
	123	Verify all Fields are Enabled		All Fields and Enabled	Result as expected	PASS	
5.2.8,5.2.8.1	124	Select RMA Status Dropdown	"Closed"	Error Pop up Displays that RMA is not 100% Complete	Result as expected	PASS	
5.2.8.1.1	125	Select OK Button		Error Pop up closes and User is returned to RMA Details Screen	Result as expected	PASS	
	126	Check Replacement/Repair Product Ship? Checkbox		Checkbox is set to Checked	Result as expected	PASS	
5.1.1	127	Verify Progress Bar displays 100%		Progress Bar Displays 100%	Result as expected	PASS	
5.2.8,5.2.8.1	128	Select RMA Status Dropdown	"Closed"	Confirmation Pop Up Displays	Result as expected	PASS	
5.2.8.1.1	129	Select Confirm Button		Confirmation Pop up closes and all field are not editable/enabled	Result as expected	PASS	
5.7	130	From RMA Details Screen Select Home Button to return to the Home Screen		Home Screen Displays	Result as expected	PASS	
	131	Verify changes made to RMA in the Table		Changes in the RMA Displays	Result as expected	PASS	
		Select "New RMA" Button and create a new random RMA	Anything	Newly created RMA displays in the Table			
2.7, 2.7.2	11	Check the Box in the "Delete?" Column in the Table for the recently created RMA		Box is marked with a check displays	Result as expected	PASS	
	12	Select Delete Button		Delete Confirmation Pop Up displays	Result as expected	PASS	
2.7.3	15	Select Confirm button		Delete Confirmation hides and User is returned to Home Screen	Result as expected	PASS	
5.2.8.2	141	Verify newly Created RMA is Deleted and removed from the Database		Newly created RMA does not display in the Table and removed from the database	Result as expected	PASS	
2.1	142	Verify total items in the list table and the last updated date and time updated correctly		Number of total item in the list table and last updated date and time displays correctly	Result as expected	PASS	
2.6	143	Select the Exit Button		Exit Confirm Pop Up displays	Result as expected	PASS	
2.6.1	144	Select No button		Exit Confirmation hides and User is returned to Home Screen	Result as expected	PASS	
	145	Select Exit Button		Exit Confirmation Pop Up displays	Result as expected	PASS	
2.6.1	146	Select Confirm button		Delete Confirmation hides and Application is closed	Result as expected	PASS	

White Box Testing

Please reference source code for White Box testing directly in the programming files.

Known Bugs and Issues

We do not have any known bugs or issues with our program.