Cartify

Generated by Doxygen 1.12.0

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 Ui Namespace Reference

Chapter 6

Class Documentation

6.1 Cart Class Reference

Represents a shopping cart that holds a collection of products.

```
#include <cart.h>
```

Public Member Functions

- Cart ()

    Constructs an empty Cart object.
- void addProduct (const Product &product)

    Adds a product to the cart.
- void removeProductById (int productId)

    Removes a product from the cart by its ID.
- const vector< Product > & getProducts () const

    Retrieves the list of products in the cart.
- void clearCart ()

    Clears all products from the cart.

Private Attributes

- vector< Product > products

    List of products in the cart.

6.1.1 Detailed Description

Represents a shopping cart that holds a collection of products.

The Cart class provides functionality to add, remove, and manage products in a shopping cart. It also allows clearing the cart and retrieving the list of products.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Cart()

```
Cart::Cart ()
```

Constructs an empty Cart object.

Default constructor for the Cart class.

6.1.3 Member Function Documentation

6.1.3.1 addProduct()

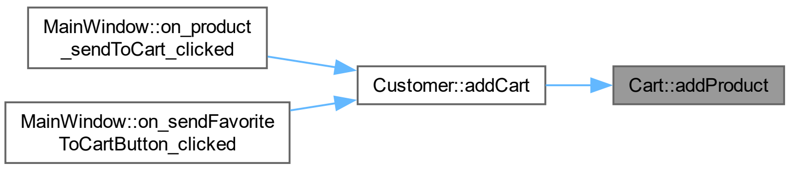
```
void Cart::addProduct (
             const Product & product)
```

Adds a product to the cart.

Parameters

product	The product to be added to the cart.

Here is the caller graph for this function:

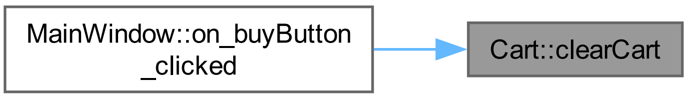

MainWindow::on_product _sendToCart_clicked
MainWindow::on_sendFavorite ToCartButton_clicked
Customer::addCart
Cart::addProduct

6.1.3.2 clearCart()

```
void Cart::clearCart ()
```

Clears all products from the cart.

Here is the caller graph for this function:



6.1.3.3 getProducts()

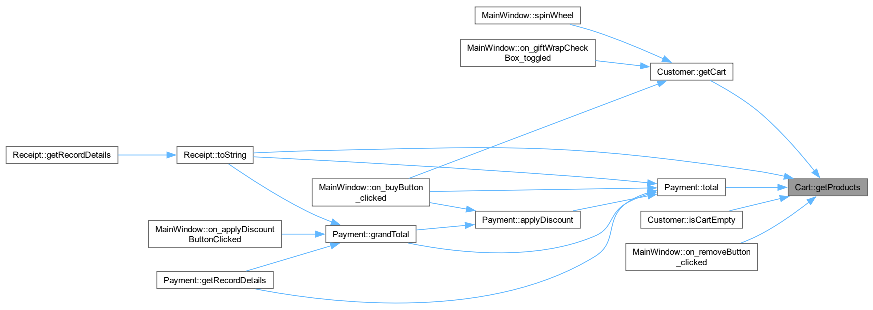
```
const std::vector< Product > & Cart::getProducts () const
```

Retrieves the list of products in the cart.

Returns

A constant reference to the vector of products in the cart.

Here is the caller graph for this function:


MainWindow::spinWheel
MainWindow::on_giftWrapCheck Box_toggled
Customer::getCart
Receipt::getRecordDetails
Receipt::toString
MainWindow::on_buyButton _clicked
Payment::total
Cart::getProducts
MainWindow::on_applyDiscount ButtonClicked
Payment::grandTotal
Payment::applyDiscount
Customer::isCartEmpty
MainWindow::on_removeButton _clicked
Payment::getRecordDetails

6.1.3.4 removeProductById()

```
void Cart::removeProductById (
                int productId)
```

Removes a product from the cart by its ID.

Parameters

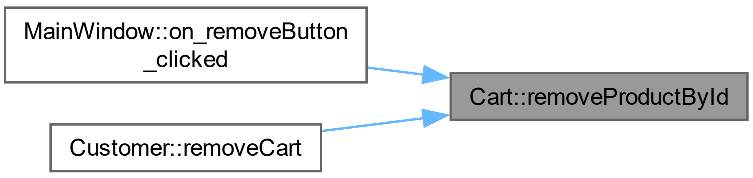
product←→ Id	The ID of the product to remove.

This function finds the product with the specified ID and removes it from the list of products in the cart.

Parameters

product←→ Id	The ID of the product to remove.

Here is the caller graph for this function:



6.1.4 Member Data Documentation

6.1.4.1 products

```
vector<Product> Cart::products  [private]
```

List of products in the cart.

The documentation for this class was generated from the following files:

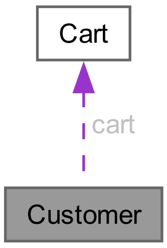
- cart.h
- cart.cpp

6.2 Customer Class Reference

Represents a customer in the Cartify system.

```
#include <customer.h>
```

Collaboration diagram for Customer:



Public Member Functions

- Customer (QString name, QString surname, ProductType productType, QString email, QString password)

    Constructs a Customer object.
- ∼Customer ()

    Destructor for the Customer class.
- Cart getCartObject ()

    Retrieves the customer's cart object.
- void addPoint (int amount)

    Adds points to the customer's account.
- int myPoints () const

    Retrieves the customer's current points.
- bool isCartEmpty (Product prod)

    Checks if the cart is empty.

- bool addFavorite (Product prod)

  Adds a product to the customer's favorites.
- void removeFavorite (int id)

  Removes a product from the customer's favorites by ID.
- QVector< Product > getFavorites ()

  Retrieves the customer's list of favorite products.
- void setFavorites (QString favorites, QVector< Product > products)

  Sets the customer's favorite products based on a string of IDs.
- void addCart (Product prod)

  Adds a product to the customer's cart.
- void removeCart (int id)

  Removes a product from the customer's cart by ID.
- vector< Product > getCart ()

  Retrieves the list of products in the customer's cart.
- bool likeProduct (int productId)

  Marks a product as liked by the customer.
- bool hasLikedProduct (int productId) const

  Checks if the customer has liked a specific product.
- void addPurchasedProduct (const Product &product)

  Adds a product to the customer's list of purchased products.
- bool hasPurchasedProduct (int productId) const

  Checks if the customer has purchased a specific product.
- QString getName () const

  Retrieves the customer's first name.
- void setName (const QString &newName)

  Sets the customer's first name.
- QString getSurname () const

  Retrieves the customer's surname.
- void setSurname (const QString &newSurname)

  Sets the customer's surname.
- QString getEmail () const

  Retrieves the customer's email address.
- void setPassword (const QString &newPassword)

  Sets the customer's password.
- ProductType getProductType () const

  Retrieves the customer's preferred product type.
- void setProductType (ProductType newProductType)

  Sets the customer's preferred product type.
- int getPoints () const

  Retrieves the customer's accumulated points.
- void setPoints (int newPoints)

  Sets the customer's accumulated points.

Public Attributes

- Cart cart

Private Attributes

- std::set< int > likedProductIds

  Set of product IDs liked by the customer.
- std::vector< Product > purchasedProducts

  List of products purchased by the customer.
- QString name

  Customer's first name.
- QString surname

  Customer's surname.
- QString email

  Customer's email address.
- QString password

  Customer's password.
- ProductType productType

  Customer's preferred product type.
- int point

  Customer's accumulated points.
- QVector< Product > favorites

  List of favorite products.
- QVector< Product > previousOrders

  List of previous orders.

6.2.1 Detailed Description

Represents a customer in the Cartify system.

The Customer class handles user-specific operations such as managing favorites, adding products to the cart, and tracking points.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Customer()

```
Customer::Customer (
              QString name,
              QString surname,
              ProductType productType,
              QString email,
              QString password)
```

Constructs a Customer object.

Constructs a Customer object with the given details.

Parameters

name	The customer's first name.
surname	The customer's surname.
productType	The customer's preferred product type.
email	The customer's email address.
password	The customer's password.

6.2.2.2 ∼Customer()

```
Customer::∼Customer ()
```

Destructor for the Customer class.

6.2.3 Member Function Documentation

6.2.3.1 addCart()

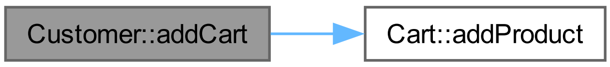
```
void Customer::addCart (
            Product prod)
```

Adds a product to the customer's cart.

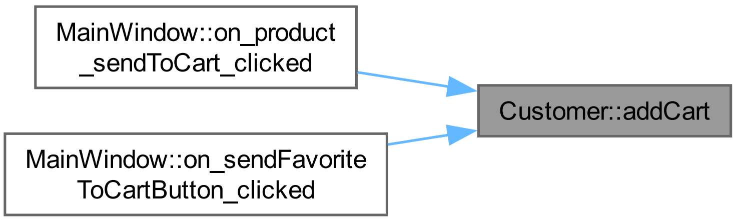
Parameters

prod	The product to add.

Here is the call graph for this function:



Here is the caller graph for this function:


MainWindow::on_product _sendToCart_clicked
MainWindow::on_sendFavorite ToCartButton_clicked
Customer::addCart

6.2.3.2 addFavorite()

```
bool Customer::addFavorite (
            Product prod)
```

Adds a product to the customer's favorites.

Adds a product to the customer's favorites if it's not already present.

Parameters

prod	The product to add.

Returns

True if the product was successfully added, false if it's already in favorites.

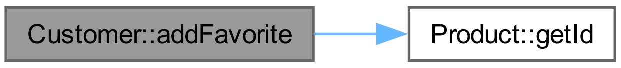
Parameters

prod	The product to add.

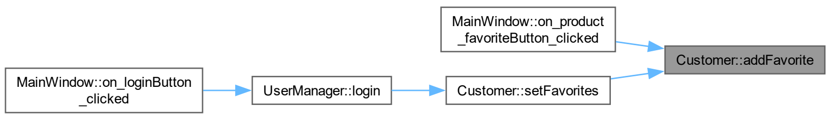
Returns

True if the product was successfully added, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:


MainWindow::on_product _favoriteButton_clicked
Customer::addFavorite
MainWindow::on_loginButton _clicked
UserManager::login
Customer::setFavorites

6.2.3.3 addPoint()

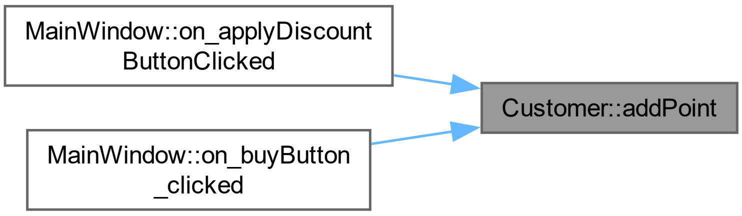
```
void Customer::addPoint (
             int amount)
```

Adds points to the customer's account.

Parameters

amount	The amount of points to add.

Here is the caller graph for this function:


MainWindow::on_applyDiscount ButtonClicked
MainWindow::on_buyButton _clicked
Customer::addPoint

6.2.3.4 addPurchasedProduct()

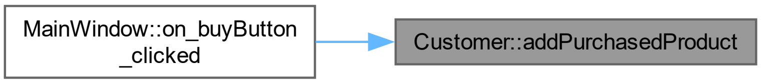
```
void Customer::addPurchasedProduct (
             const Product & product)
```

Adds a product to the customer's list of purchased products.

Parameters

product	The product to add.

Here is the caller graph for this function:


MainWindow::on_buyButton _clicked
Customer::addPurchasedProduct

6.2.3.5 getCart()

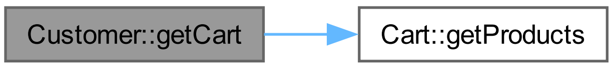
```
vector< Product > Customer::getCart ()
```

Retrieves the list of products in the customer's cart.

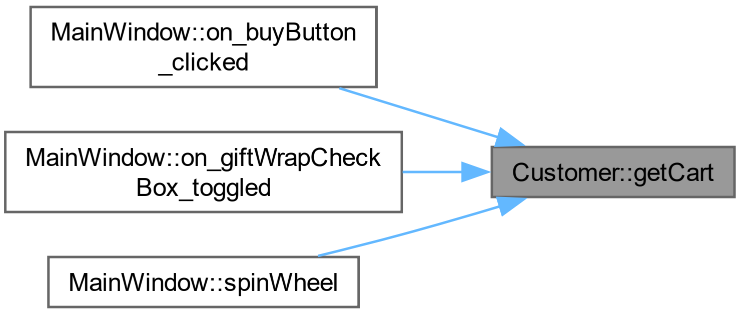
Returns

A vector of products currently in the cart.

Here is the call graph for this function:



Here is the caller graph for this function:


MainWindow::on_buyButton _clicked
MainWindow::on_giftWrapCheck Box_toggled
MainWindow::spinWheel
Customer::getCart

6.2.3.6 getCartObject()

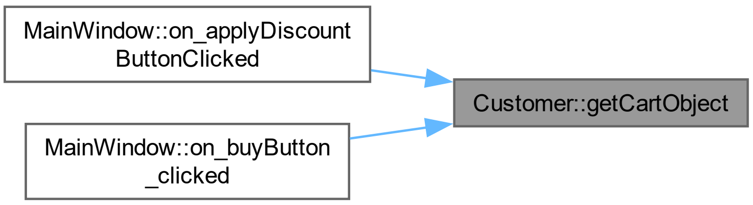
```
Cart Customer::getCartObject ()
```

Retrieves the customer's cart object.

Returns

A Cart object representing the customer's shopping cart.

Here is the caller graph for this function:


MainWindow::on_applyDiscount ButtonClicked
MainWindow::on_buyButton _clicked
Customer::getCartObject

6.2.3.7 getEmail()

```
QString Customer::getEmail () const  [inline]
```

Retrieves the customer's email address.

Returns

A QString representing the customer's email address.

6.2.3.8 getFavorites()

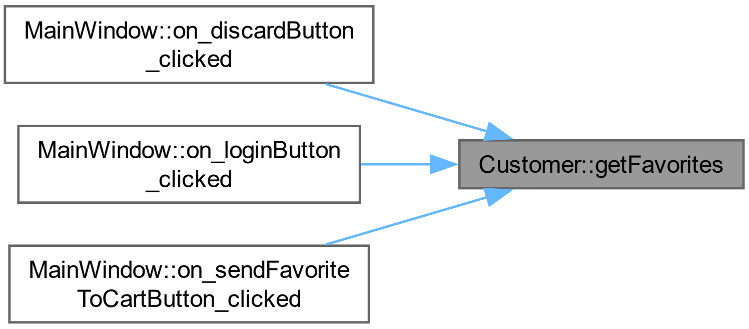
```
QVector< Product > Customer::getFavorites ()
```

Retrieves the customer's list of favorite products.

Returns

A QVector containing the customer's favorite products.

Here is the caller graph for this function:


MainWindow::on_discardButton _clicked
MainWindow::on_loginButton _clicked
MainWindow::on_sendFavorite ToCartButton_clicked
Customer::getFavorites

6.2.3.9 getName()

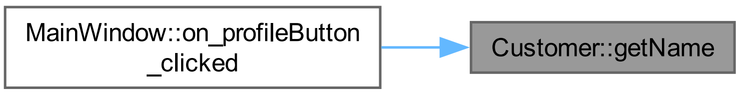
```
QString Customer::getName () const  [inline]
```

Retrieves the customer's first name.

Returns

A QString representing the customer's first name.

Here is the caller graph for this function:



6.2.3.10 getPoints()

```
int Customer::getPoints () const  [inline]
```

Retrieves the customer's accumulated points.

Returns

An integer representing the customer's points.

6.2.3.11 getProductType()

```
ProductType Customer::getProductType () const  [inline]
```

Retrieves the customer's preferred product type.

Returns

The ProductType representing the customer's preference.

6.2.3.12 getSurname()

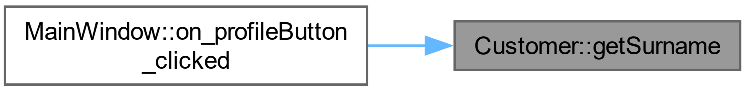
```
QString Customer::getSurname () const  [inline]
```

Retrieves the customer's surname.

Returns

A QString representing the customer's surname.

Here is the caller graph for this function:



6.2.3.13 hasLikedProduct()

```
bool Customer::hasLikedProduct (
             int productId) const
```

Checks if the customer has liked a specific product.

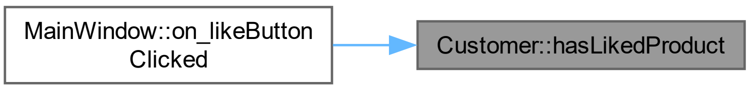
Parameters

product←Id	The ID of the product to check.

Returns

True if the product is liked, false otherwise.

Here is the caller graph for this function:



6.2.3.14 hasPurchasedProduct()

```
bool Customer::hasPurchasedProduct (
             int productId) const
```

Checks if the customer has purchased a specific product.

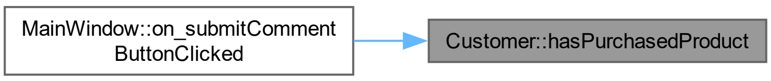
Parameters

product↩ Id	The ID of the product to check.

Returns

True if the product has been purchased, false otherwise.

Here is the caller graph for this function:


MainWindow::on_submitComment ButtonClicked
Customer::hasPurchasedProduct

6.2.3.15 isCartEmpty()

```
bool Customer::isCartEmpty (
            Product prod)
```

Checks if the cart is empty.

Parameters

prod	A product to check against.

Returns

True if the cart is empty, false otherwise.

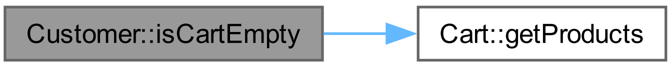
Parameters

prod	A product to check against (unused in logic).

Returns

True if the cart is empty, false otherwise.

Here is the call graph for this function:



6.2.3.16 likeProduct()

```
bool Customer::likeProduct (
             int productId)
```

Marks a product as liked by the customer.

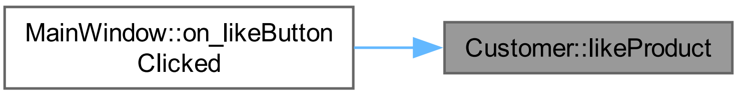
Parameters

product←↩ Id	The ID of the product to like.

Returns

True if the product was successfully liked, false if it was already liked.

Here is the caller graph for this function:



6.2.3.17 myPoints()

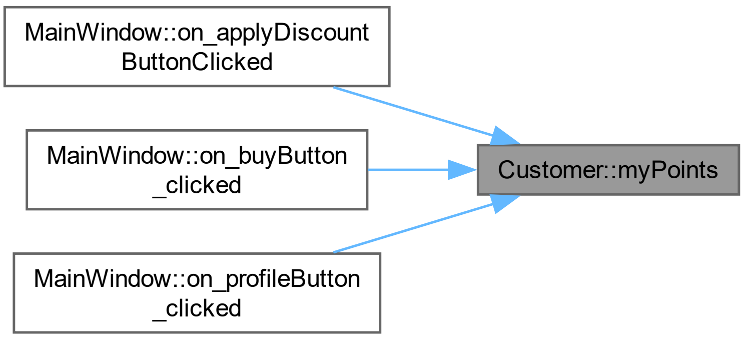
```
int Customer::myPoints () const
```

Retrieves the customer's current points.

Returns

The number of points the customer has.

Here is the caller graph for this function:


MainWindow::on_applyDiscount ButtonClicked
MainWindow::on_buyButton _clicked
MainWindow::on_profileButton _clicked
Customer::myPoints

6.2.3.18 removeCart()

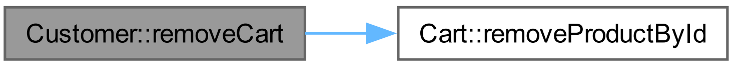
```
void Customer::removeCart (
             int id)
```

Removes a product from the customer's cart by ID.

Parameters

id	The ID of the product to remove.

Here is the call graph for this function:



6.2.3.19 removeFavorite()

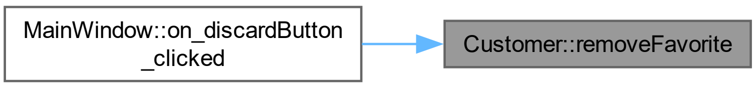
```
void Customer::removeFavorite (
             int id)
```

Removes a product from the customer's favorites by ID.

Parameters

id	The ID of the product to remove.

Here is the caller graph for this function:



6.2.3.20 setFavorites()

```
void Customer::setFavorites (
             QString favorites,
             QVector< Product > products)
```

Sets the customer's favorite products based on a string of IDs.

Parameters

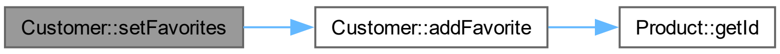
favorites	A comma-separated string of product IDs.
products	A QVector of available products.

Parses a comma-separated string of product IDs and adds matching products to the customer's favorites.

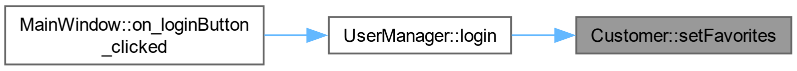
Parameters

favorites	A comma-separated string of product IDs.
products	A QVector of available products.

Here is the call graph for this function:



Here is the caller graph for this function:



6.2.3.21 setName()

```
void Customer::setName (
             const QString & newName)  [inline]
```

Sets the customer's first name.

Parameters

newName	The new name to set for the customer.

6.2.3.22 setPassword()

```
void Customer::setPassword (
             const QString & newPassword)  [inline]
```

Sets the customer's password.

Parameters

newPassword	The new password to set for the customer.

6.2.3.23 setPoints()

```
void Customer::setPoints (
             int newPoints)  [inline]
```

Sets the customer's accumulated points.

Parameters

newPoints	The new points value to set for the customer.

6.2.3.24 setProductType()

```
void Customer::setProductType (
             ProductType newProductType)  [inline]
```

Sets the customer's preferred product type.

Parameters

newProductType	The new product type to set for the customer.

6.2.3.25 setSurname()

```
void Customer::setSurname (
             const QString & newSurname)  [inline]
```

Sets the customer's surname.

Parameters

newSurname	The new surname to set for the customer.

6.2.4 Member Data Documentation

6.2.4.1 cart

```
Cart Customer::cart
```

6.2.4.2 email

```
QString Customer::email  [private]
```

Customer's email address.

6.2.4.3 favorites

```
QVector<Product> Customer::favorites  [private]
```

List of favorite products.

6.2.4.4 likedProductIds

```
std::set<int> Customer::likedProductIds  [private]
```

Set of product IDs liked by the customer.

6.2.4.5 name

```
QString Customer::name  [private]
```

Customer's first name.

6.2.4.6 password

```
QString Customer::password  [private]
```

Customer's password.

6.2.4.7 point

```
int Customer::point  [private]
```

Customer's accumulated points.

6.2.4.8 previousOrders

```
QVector<Product> Customer::previousOrders  [private]
```

List of previous orders.

6.2.4.9 productType

```
ProductType Customer::productType  [private]
```

Customer's preferred product type.

6.2.4.10 purchasedProducts

```
std::vector<Product> Customer::purchasedProducts  [private]
```

List of products purchased by the customer.

6.2.4.11 surname

```
QString Customer::surname  [private]
```

Customer's surname.

The documentation for this class was generated from the following files:

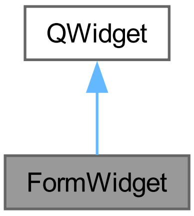
- customer.h
- customer.cpp

6.3 FormWidget Class Reference

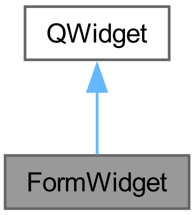
A widget that provides a login and sign-up form interface.

```
#include <FormWidget.h>
```

Inheritance diagram for FormWidget:



Collaboration diagram for FormWidget:



Public Member Functions

- FormWidget (QWidget ∗parent=nullptr)

  Constructs a FormWidget object.

Private Attributes

- QLabel ∗ emailLabel

  Label for the email field.
- QLineEdit ∗ emailField

  Input field for the user's email.
- QLabel ∗ passwordLabel

  Label for the password field.
- QLineEdit ∗ passwordField

  Input field for the user's password.
- QPushButton ∗ loginButton

  Button for logging in.
- QPushButton ∗ signupButton

  Button for signing up.
- QGridLayout ∗ formLayout

  Layout manager for the form elements.

6.3.1 Detailed Description

A widget that provides a login and sign-up form interface.

This widget includes fields for email and password input, as well as buttons for logging in and signing up. It uses a grid layout for organization.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 FormWidget()

```
FormWidget::FormWidget (
            QWidget * parent = nullptr)  [explicit]
```

Constructs a FormWidget object.

Constructs the FormWidget and initializes the UI elements.

Parameters

parent	The parent widget. Defaults to nullptr.

The FormWidget provides a simple login and sign-up form. It includes labels and fields for email and password input, as well as buttons for login and sign-up actions.

Parameters

parent	The parent widget. Defaults to nullptr.

6.3.3 Member Data Documentation

6.3.3.1 emailField

```
QLineEdit* FormWidget::emailField  [private]
```

Input field for the user's email.

6.3.3.2 emailLabel

```
QLabel* FormWidget::emailLabel  [private]
```

Label for the email field.

6.3.3.3 formLayout

```
QGridLayout* FormWidget::formLayout  [private]
```

Layout manager for the form elements.

6.3.3.4 loginButton

```
QPushButton* FormWidget::loginButton  [private]
```

Button for logging in.

6.3.3.5 passwordField

```
QLineEdit* FormWidget::passwordField  [private]
```

Input field for the user's password.

6.3.3.6 passwordLabel

```
QLabel* FormWidget::passwordLabel  [private]
```

Label for the password field.

6.3.3.7 signupButton

```
QPushButton* FormWidget::signupButton  [private]
```

Button for signing up.

The documentation for this class was generated from the following files:

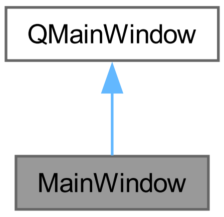
- FormWidget.h
- FormWidget.cpp

6.4 MainWindow Class Reference

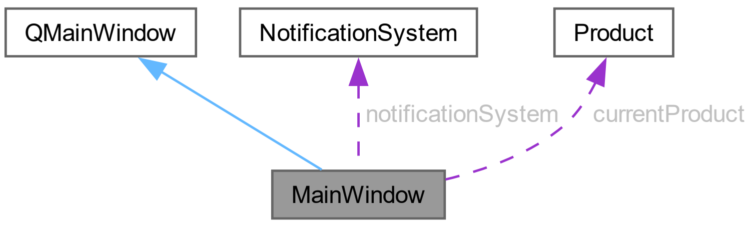
Main application window for Cartify.

```
#include <mainwindow.h>
```

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



Public Member Functions

- MainWindow (QWidget ∗parent=nullptr)

  Constructs the MainWindow object and initializes the GUI.
- ∼MainWindow ()

  Destroys the MainWindow object and performs cleanup.

Protected Member Functions

- void closeEvent (QCloseEvent ∗event) override

  Handles the close event to perform necessary cleanup.

Private Slots

- void spinWheel ()

  Spins the discount wheel.
- void applyDiscount (int percentage)

  Applies a discount to the current purchase.
- void on_searchButton_clicked ()

  Handles the search button click event.
- void on_applyDiscountButtonClicked ()

  Handles the apply discount button click event.
- void on_productButtonClicked (Product &product)

  Handles product selection when a product button is clicked.
- void on_productButton_clicked ()

  Detects the product button click event and processes it.
- void on_likeButtonClicked ()

  Handles the like button click event for a product.
- void on_submitCommentButtonClicked ()

  Submits a comment for the selected product.
- void displayCommentsForProduct (const Product &product)

  Displays comments for the given product.
- void on_gotoSignUp_clicked ()

  Navigates to the sign-up page.
- void on_signUpButton_clicked ()

  Handles user sign-up.
- void on_loginButton_clicked ()

  Handles user login.
- void on_electronicsButton_clicked ()

  Displays the electronics product page.
- void on_clothesButton_clicked ()

  Displays the clothes product page.
- void on_mainScreenButton_clicked ()

  Navigates back to the main screen.
- void on_logoutButton_clicked ()

  Logs out the current user.
- void on_product_favoriteButton_clicked ()

  Adds the selected product to the user's favorites.
- void on_product_sendToCart_clicked ()

  Sends the selected product to the user's shopping cart.

- void on_profileButton_clicked ()

    Displays the user's profile page.
- void on_discardButton_clicked ()

    Removes a product from the user's favorites.
- void on_sendFavoriteToCartButton_clicked ()

    Sends a favorite product to the shopping cart.
- void on_cartButton_clicked ()

    Displays the cart page.
- void on_buyButton_clicked ()

    Processes the purchase of items in the cart.
- void on_removeButton_clicked ()

    Removes an item from the cart.
- void on_sizeComboBox_currentTextChanged (const QString &arg1)

    Updates the selected size of a product.
- void on_pass_to_loginPage_clicked ()

    Navigates to the login page.
- void setProductButton (QPushButton ∗button, const QString &picturePath)

    Sets the product button with the given picture.
- void setupProductButtons ()

    Initializes product buttons.
- void showDiscountColors ()

    Displays discount color codes on the GUI.
- void on_giftWrapCheckBox_toggled (bool checked)

    Handles the toggling of the gift wrap checkbox.

Private Member Functions

- void connectProductButtons ()

    Private helper functions.

Private Attributes

- Ui::MainWindow ∗ ui

    The UI object for the MainWindow.
- QListWidget ∗ commentListWidget

    Widget for displaying comments.
- NotificationSystem ∗ notificationSystem

    Handles user notifications.
- QVector< Product > products

    List of products displayed in the application.
- QVector< QPushButton ∗ > productButtons

    Buttons for product selection.
- QMap< QPushButton ∗, Product ∗ > buttonProductMap

    Maps buttons to corresponding products.
- Product ∗ currentProduct

    The currently selected product.
- QGraphicsScene ∗ scene

    Graphics and discount wheel elements.
- QGraphicsEllipseItem ∗ wheel
- QTimer ∗ timer

- int currentAngle
- int targetAngle
- QStringList wheelRewards
- double currentSpeed
- double minSpeed
- double deceleration
- QList< QGraphicsPathItem * > slices
- QMap< int, QString > colorMap
- QMap< QString, int > discountMap
- QTimer * blinkTimer
- int blinkCount
- bool blinkState
- int chosenSliceIndex
- int currentDiscount
- bool discountApplied
- const double giftWrapFee = 10.0

  Constants.

6.4.1 Detailed Description

Main application window for Cartify.

Represents the main window of the application and manages the user interface.

The MainWindow class provides the primary GUI for the Cartify system, handling user interaction, product management, shopping cart, and discount functionality. It also manages user authentication and facilitates the purchase process.

The MainWindow class is responsible for initializing and managing the graphical user interface (GUI), as well as handling application logic such as notifications, animations, and user interactions.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 MainWindow()

```
MainWindow::MainWindow (
            QWidget * parent = nullptr)
```

Constructs the MainWindow object and initializes the GUI.

Parameters

parent	The parent widget. Defaults to nullptr.

This constructor sets up the main window, initializes various components like the notification system, the graphics scene, and timers, and prepares the application state.

Parameters

parent	The parent widget. Defaults to nullptr.

6.4.2.2 ∼MainWindow()

```
MainWindow::∼MainWindow ()
```

Destroys the MainWindow object and performs cleanup.

Destructor for the MainWindow class.

This method releases dynamically allocated resources, including the user interface object and the notification system instance, to prevent memory leaks.

6.4.3 Member Function Documentation

6.4.3.1 applyDiscount

```
void MainWindow::applyDiscount (
            int percentage)  [private], [slot]
```

Applies a discount to the current purchase.

Parameters

percentage	The percentage of the discount to apply.

This method sets the discount percentage for the current purchase, updates the `currentDiscount` variable, and displays an informational message to the user.

Parameters

percentage	The percentage value of the discount to be applied.

Note

The `discountApplied` flag is set to true to indicate that a discount has been applied.

6.4.3.2 closeEvent()

```
void MainWindow::closeEvent (
            QCloseEvent * event)  [override], [protected]
```

Handles the close event to perform necessary cleanup.

Handles the close event for the main window.

This function logs out the user and ensures proper cleanup of resources when the application window is closed.

Parameters

event	The QCloseEvent triggered when the window is closed.

This method is triggered when the main window is about to close. It ensures that the user is logged out properly by calling the `on_logoutButton_clicked` method before the application exits.

Parameters

event	Pointer to the `QCloseEvent` object that contains information about the close event.

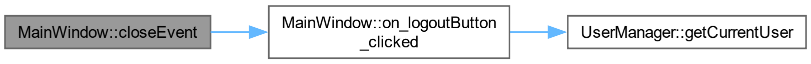
Note

This ensures that any necessary cleanup or state reset associated with user logout is performed before the application is terminated.

See also

MainWindow::on_logoutButton_clicked

Here is the call graph for this function:


MainWindow::closeEvent
MainWindow::on_logoutButton _clicked
UserManager::getCurrentUser

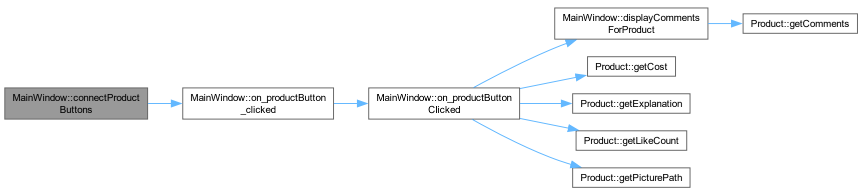
6.4.3.3 connectProductButtons()

```
void MainWindow::connectProductButtons ()  [private]
```

Private helper functions.

Connects product buttons to their click event handler.

This method iterates over all product buttons and connects their `clicked` signal to the `on_productButton↵_clicked` slot. This enables the application to respond to user interactions with the product buttons. Here is the call graph for this function:


MainWindow::connectProduct Buttons
MainWindow::on_productButton _clicked
MainWindow::on_productButton Clicked
MainWindow::displayComments ForProduct
Product::getComments
Product::getCost
Product::getExplanation
Product::getLikeCount
Product::getPicturePath

6.4.3.4 displayCommentsForProduct

```
void MainWindow::displayCommentsForProduct (
            const Product & product)  [private], [slot]
```

Displays comments for the given product.

Displays the comments for the specified product in the UI.

Updates the comment list in the GUI to show comments for the selected product.

Parameters

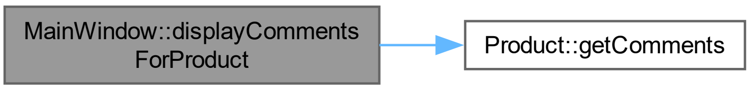
product	The product whose comments are displayed.

This method retrieves the list of comments associated with the given product and populates the comment list widget with them. Existing comments in the widget are cleared before adding the new comments.

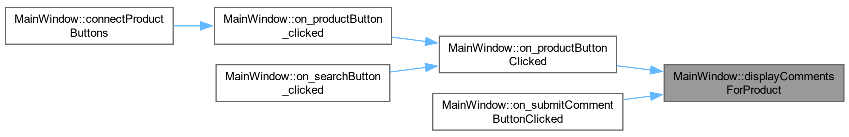
Parameters

product	The `Product` object whose comments are to be displayed.

Here is the call graph for this function:



Here is the caller graph for this function:


MainWindow::connectProduct Buttons
MainWindow::on_productButton _clicked
MainWindow::on_searchButton _clicked
MainWindow::on_productButton Clicked
MainWindow::on_submitComment ButtonClicked
MainWindow::displayComments ForProduct

6.4.3.5 on_applyDiscountButtonClicked

```
void MainWindow::on_applyDiscountButtonClicked ()  [private], [slot]
```

Handles the apply discount button click event.

Handles the "Apply Discount" button click event.

Validates and applies a discount based on user input.

This method validates the entered discount code and checks the user's points to determine if the discount can be applied. If the code is valid and the user has sufficient points, the discount is applied to the payment and the user's points are updated. The grand total is recalculated and displayed to the user.

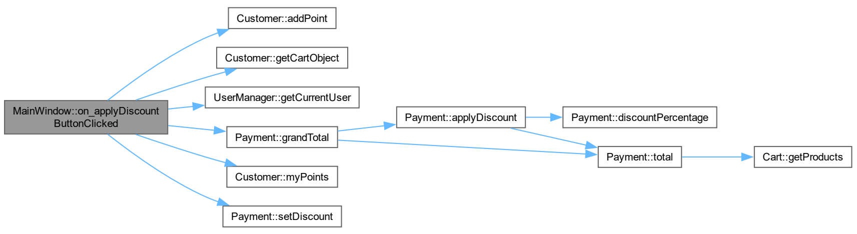
Note

The discount codes and their required points are as follows:

- D50: 50% discount, requires 300 points
- D20: 20% discount, requires 200 points
- D10: 10% discount, requires 100 points

If the discount code is invalid or the user does not have enough points, appropriate warning messages are displayed.

Here is the call graph for this function:



6.4.3.6 on_buyButton_clicked

```
void MainWindow::on_buyButton_clicked () [private], [slot]
```

Processes the purchase of items in the cart.

Handles the click event for the "Buy" button.

Validates payment and completes the purchase process.

This method processes the user's purchase by performing the following actions:

- Validates the selected payment method and ensures the cart is not empty.
- Handles gift wrap and shipping fees if applicable.
- Applies discounts based on the user's points or entered discount code.
- Calculates the final grand total after applying discounts and additional fees.
- Generates a receipt in HTML format and displays it in the receipt browser.
- Clears the user's cart and updates the purchases combo box.
- Resets the discount state and navigates to the receipt page.

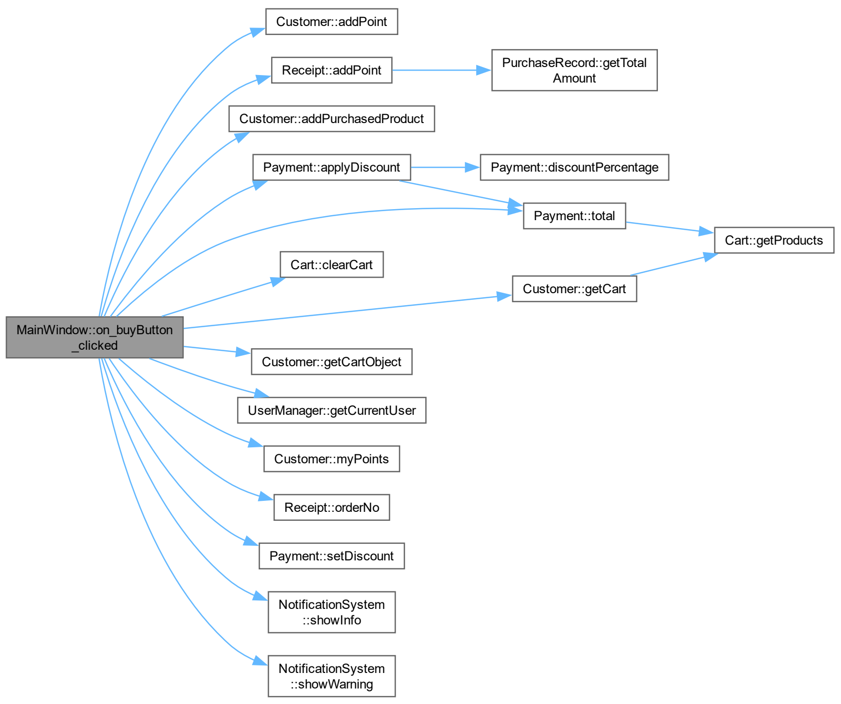
Note

If any validation fails (e.g., missing payment method or invalid discount code), appropriate warning or informational messages are displayed.

See also

Payment, Receipt

Here is the call graph for this function:


Customer::addPoint
Receipt::addPoint
PurchaseRecord::getTotalAmount
Customer::addPurchasedProduct
Payment::applyDiscount
Payment::discountPercentage
Payment::total
Cart::getProducts
Cart::clearCart
Customer::getCart
MainWindow::on_buyButton_clicked
Customer::getCartObject
UserManager::getCurrentUser
Customer::myPoints
Receipt::orderNo
Payment::setDiscount
NotificationSystem::showInfo
NotificationSystem::showWarning

6.4.3.7 on_cartButton_clicked

```
void MainWindow::on_cartButton_clicked ()  [private], [slot]
```

Displays the cart page.

Handles the click event for the "Cart" button.

This method navigates the user to the cart page by changing the current index of the stacked widget.

Note

This function assumes that the cart page is at index 3 of the stacked widget.

6.4.3.8 on_clothesButton_clicked

```
void MainWindow::on_clothesButton_clicked ()  [private], [slot]
```

Displays the clothes product page.

Handles the click event for the "Clothes" button.

This method navigates the user to the clothes product page and adjusts the visibility of UI components specific to the clothes category.

- Electronics-related filters and labels are hidden.
- Clothes-related filters and labels are made visible.

6.4.3.9 on_discardButton_clicked

```
void MainWindow::on_discardButton_clicked ()  [private], [slot]
```

Removes a product from the user's favorites.

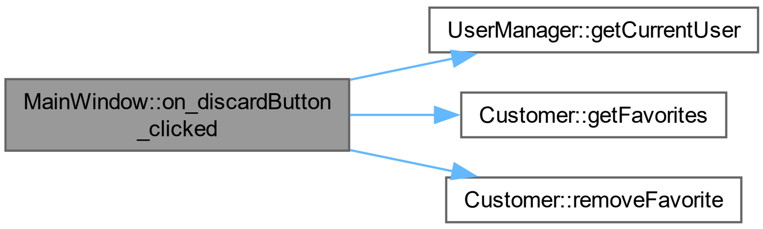
Handles the click event for the "Discard" button.

This method removes the currently selected favorite product from the user's favorites list and updates the favorites combo box to reflect the change.

Note

The product is identified by matching its explanation text with the current selection in the combo box. Once found, it is removed from both the user's favorites and the combo box.

Here is the call graph for this function:


UserManager::getCurrentUser
MainWindow::on_discardButton _clicked
Customer::getFavorites
Customer::removeFavorite

6.4.3.10 on_electronicsButton_clicked

```
void MainWindow::on_electronicsButton_clicked ()  [private], [slot]
```

Displays the electronics product page.

Handles the click event for the "Electronics" button.

This method navigates the user to the electronics product page and adjusts the visibility of UI components specific to the electronics category.

- Clothes-related filters and labels are hidden.
- Electronics-related filters and labels are made visible.

6.4.3.11 on_giftWrapCheckBox_toggled

```
void MainWindow::on_giftWrapCheckBox_toggled (
            bool checked)  [private], [slot]
```

Handles the toggling of the gift wrap checkbox.

Ensures that the checkbox state is valid and updates the GUI accordingly.

Parameters

checked	The state of the checkbox (true if checked).

This method checks if the user has selected the gift wrap option. If the checkbox is checked but the cart is empty, it displays a warning message and resets the checkbox to its unchecked state.

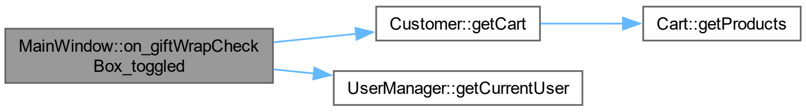
Parameters

checked	Indicates whether the checkbox is checked (true) or unchecked (false).

Note

The checkbox is blocked temporarily to prevent signal loops when resetting its state.

Here is the call graph for this function:


MainWindow::on_giftWrapCheck Box_toggled
Customer::getCart
Cart::getProducts
UserManager::getCurrentUser

6.4.3.12 on_gotoSignUp_clicked

```
void MainWindow::on_gotoSignUp_clicked ()  [private], [slot]
```

Navigates to the sign-up page.

This method switches the current view in the stacked widget to the sign-up page when the "Go to Sign Up" button is clicked.

6.4.3.13 on_likeButtonClicked

```
void MainWindow::on_likeButtonClicked ()  [private], [slot]
```

Handles the like button click event for a product.

Handles the click event for the "like" button.

Likes the currently selected product and updates the GUI.

This method allows the current user to "like" the currently selected product. If the product has already been liked by the user, an informational notification is displayed. Otherwise, the product's like count is updated, and the "like" button icon changes to indicate the action.

Note

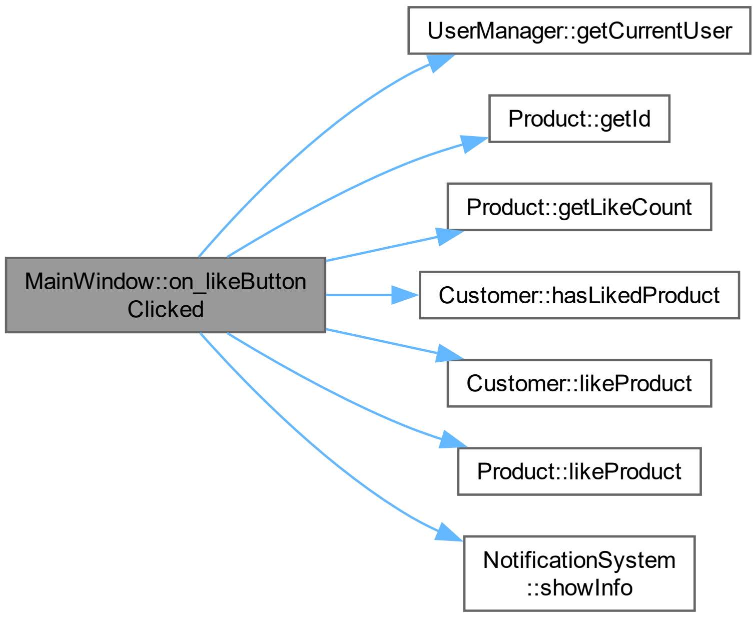
The method ensures a valid product is selected before proceeding with the action.

See also

Customer::hasLikedProduct

Product::likeProduct

Here is the call graph for this function:


UserManager::getCurrentUser
Product::getId
Product::getLikeCount
MainWindow::on_likeButton Clicked
Customer::hasLikedProduct
Customer::likeProduct
Product::likeProduct
NotificationSystem ::showInfo

6.4.3.14 on_loginButton_clicked

```
void MainWindow::on_loginButton_clicked ()  [private], [slot]
```

Handles user login.

Handles the click event for the "Login" button.

Validates credentials and logs the user in if valid.

This method attempts to log the user in using the provided email and password. If the login is successful:

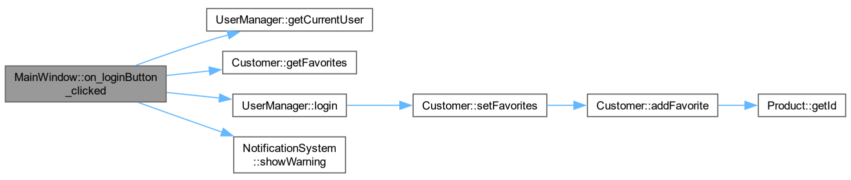
- The current user's favorites are retrieved and populated in the favorites combo box.
- The UI is navigated to the main application page.

If the login fails, a warning is displayed, and the input fields for email and password are cleared.

Note

This method uses the `userManager` to handle login logic and the `notificationSystem` to display warnings.

Here is the call graph for this function:



6.4.3.15 on_logoutButton_clicked

`void MainWindow::on_logoutButton_clicked () [private], [slot]`

Logs out the current user.

Handles the click event for the "Logout" button.

Clears user data and navigates to the login page.

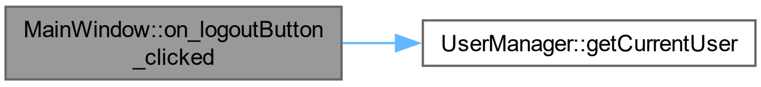
This method logs the user out by performing the following actions:

- Navigates the UI back to the login page.
- Clears the current user's favorites combo box.
- Clears the email and password input fields on the login page.

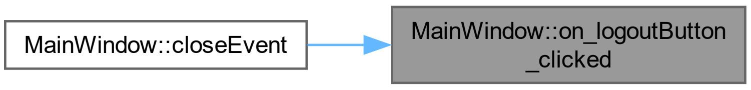
Note

The current user's session data is reset to ensure a clean state for the next login.

Here is the call graph for this function:


MainWindow::on_logoutButton _clicked
UserManager::getCurrentUser

Here is the caller graph for this function:



6.4.3.16 on_mainScreenButton_clicked

```
void MainWindow::on_mainScreenButton_clicked ()  [private], [slot]
```

Navigates back to the main screen.

Handles the click event for the "Main Screen" button.

This method resets the "like" button icon to its default (not liked) state and navigates the user back to the main screen of the application.

Note

This method assumes that the "like" button icon needs to be reset whenever the user navigates back to the main screen.

6.4.3.17 on_pass_to_loginPage_clicked

```
void MainWindow::on_pass_to_loginPage_clicked ()  [private], [slot]
```

Navigates to the login page.

Handles the click event to navigate to the login page.

This method changes the current page of the stacked widget to display the login page.

Note

This function assumes that the login page is at index 0 of the stacked widget.

6.4.3.18 on_product_favoriteButton_clicked

```
void MainWindow::on_product_favoriteButton_clicked ()  [private], [slot]
```

Adds the selected product to the user's favorites.

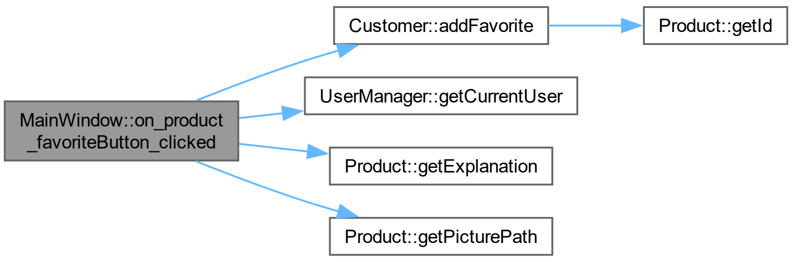
Handles the click event for the "Favorite Product" button.

This method adds the currently selected product to the user's list of favorites. If the product is successfully added as a favorite, it is also appended to the favorites combo box with its icon and description.

Note

The current user is retrieved from the `userManager` to ensure the favorite action is user-specific.

Here is the call graph for this function:



6.4.3.19 on_product_sendToCart_clicked

```
void MainWindow::on_product_sendToCart_clicked ()  [private], [slot]
```

Sends the selected product to the user's shopping cart.

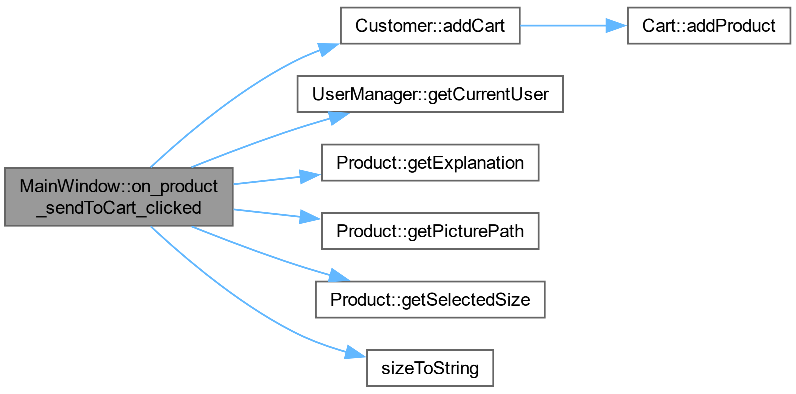
Handles the click event for the "Send to Cart" button.

This method adds the currently selected product to the user's cart and updates the purchases combo box with the product's icon, explanation, and selected size.

Note

The product's size is converted to a string using the `sizeToString` function before appending it to the combo box display.

Here is the call graph for this function:



6.4.3.20 on_productButton_clicked

```
void MainWindow::on_productButton_clicked ()  [private], [slot]
```

Detects the product button click event and processes it.

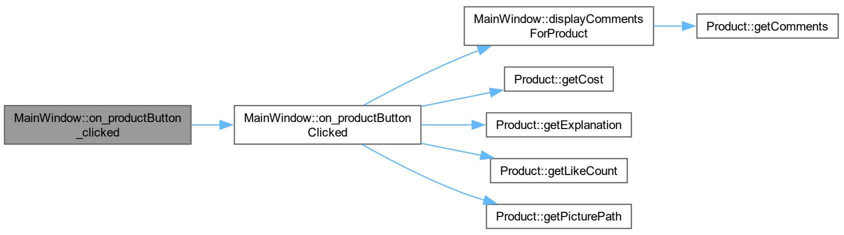
Handles the click event for a product button.

This method identifies which product button was clicked by the user, retrieves the corresponding `Product` object from the `buttonProductMap`, and triggers the `on_productButtonClicked` method with the selected product.

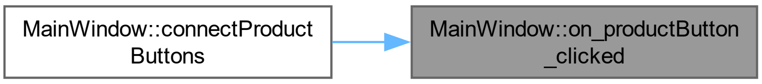
Note

The method uses the `sender()` function to determine the clicked button and ensures it exists in the `buttonProductMap` before proceeding.

Here is the call graph for this function:


MainWindow::on_productButton_clicked
MainWindow::on_productButtonClicked
MainWindow::displayCommentsForProduct
Product::getComments
Product::getCost
Product::getExplanation
Product::getLikeCount
Product::getPicturePath

Here is the caller graph for this function:


MainWindow::connectProductButtons
MainWindow::on_productButton_clicked

6.4.3.21 on_productButtonClicked

```
void MainWindow::on_productButtonClicked (
            Product & product)  [private], [slot]
```

Handles product selection when a product button is clicked.

Handles the selection of a product and updates the UI accordingly.

Parameters

product	The product associated with the clicked button.

This method is called when a product button is clicked. It updates the current product details in the UI, including the product explanation, image, cost, like count, and comments. The UI page is switched to display detailed information about the selected product.

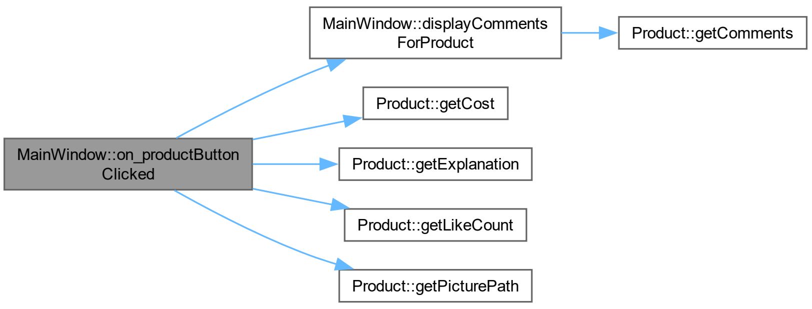
Parameters

product	Reference to the selected `Product` object.

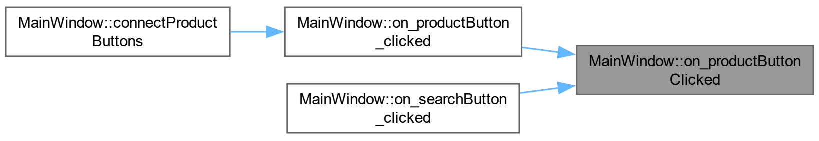
Note

The product image is scaled to fit the designated UI element while maintaining its aspect ratio.

Here is the call graph for this function:



Here is the caller graph for this function:


MainWindow::connectProduct Buttons
MainWindow::on_productButton _clicked
MainWindow::on_productButton Clicked
MainWindow::on_searchButton _clicked

6.4.3.22 on_profileButton_clicked

```
void MainWindow::on_profileButton_clicked ()  [private], [slot]
```

Displays the user's profile page.

Handles the click event for the "Profile" button.

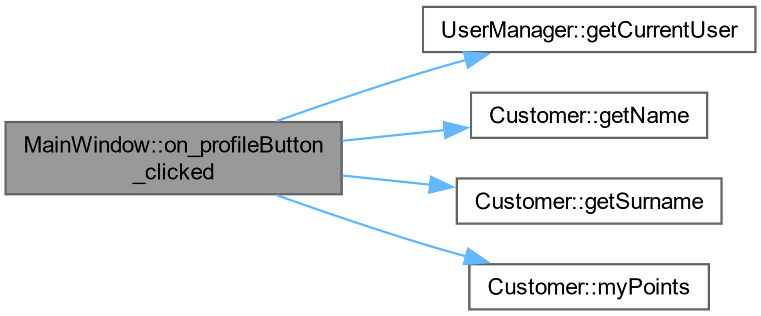
Updates the profile page with the user's information.

This method navigates the user to the profile page and updates the UI to display the current user's name, surname, and shopping credits.

Note

The `welcomeLabel` and `pointsLabel` are updated with user-specific data retrieved from the `user↩ Manager`.

Here is the call graph for this function:


MainWindow::on_profileButton _clicked
UserManager::getCurrentUser
Customer::getName
Customer::getSurname
Customer::myPoints

6.4.3.23 on_removeButton_clicked

```
void MainWindow::on_removeButton_clicked () [private], [slot]
```

Removes an item from the cart.

Handles the click event for the "Remove" button.

This method removes the currently selected product from the user's cart and updates the purchases combo box. If the cart is already empty, an informational message is displayed to the user.

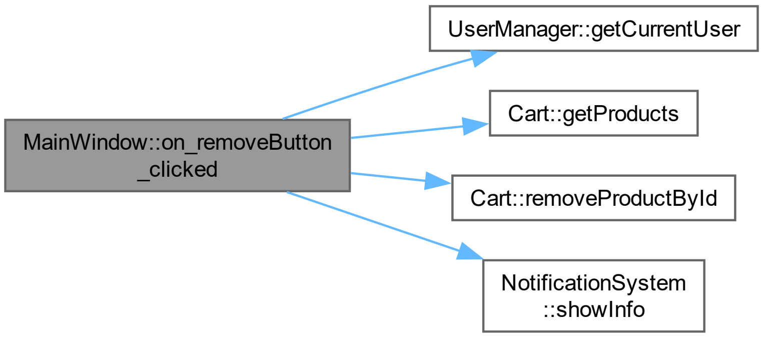
Note

The product to be removed is identified by its index in the combo box and removed both from the combo box and the user's cart.

See also

Customer::cart

Here is the call graph for this function:


UserManager::getCurrentUser
Cart::getProducts
MainWindow::on_removeButton _clicked
Cart::removeProductById
NotificationSystem ::showInfo

6.4.3.24 on_searchButton_clicked

```
void MainWindow::on_searchButton_clicked ()  [private], [slot]
```

Handles the search button click event.

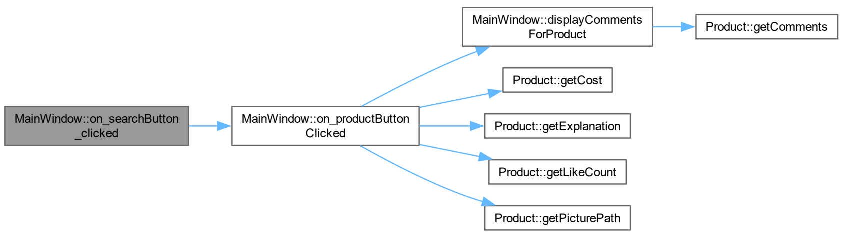
Searches for products based on the user's input and updates the GUI with the results.

This method retrieves the search query from the input field, normalizes it by trimming spaces, and searches for matching products in the product list. If a match is found, the corresponding product button is triggered. If no match is found, an informational message is displayed.

Note

If the search query is empty, the user is prompted to enter a valid search term.

Here is the call graph for this function:


MainWindow::displayComments ForProduct
Product::getComments
Product::getCost
MainWindow::on_searchButton _clicked
MainWindow::on_productButton Clicked
Product::getExplanation
Product::getLikeCount
Product::getPicturePath

6.4.3.25 on_sendFavoriteToCartButton_clicked

```
void MainWindow::on_sendFavoriteToCartButton_clicked ()  [private], [slot]
```

Sends a favorite product to the shopping cart.

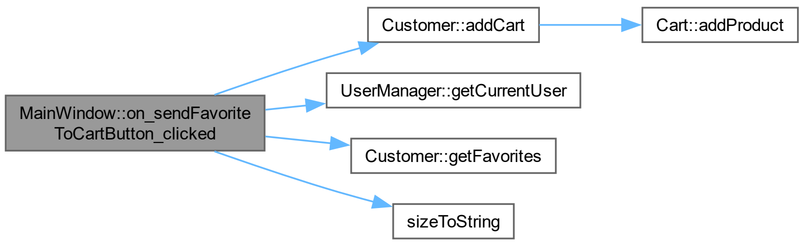
Handles the click event for the "Send Favorite to Cart" button.

This method adds the currently selected favorite product to the user's cart and updates the purchases combo box with the product's icon, explanation, and selected size.

Note

The product is identified by matching its explanation text with the current selection in the favorites combo box. Once found, it is added to the cart and the purchases combo box.

Here is the call graph for this function:


Customer::addCart
Cart::addProduct
UserManager::getCurrentUser
MainWindow::on_sendFavoriteToCartButton_clicked
Customer::getFavorites
sizeToString

6.4.3.26 on_signUpButton_clicked

```
void MainWindow::on_signUpButton_clicked ()  [private], [slot]
```

Handles user sign-up.

Handles the click event for the "Sign Up" button.

Validates user input and registers a new user if valid.

This method validates the input fields for the sign-up form. The following checks are performed:

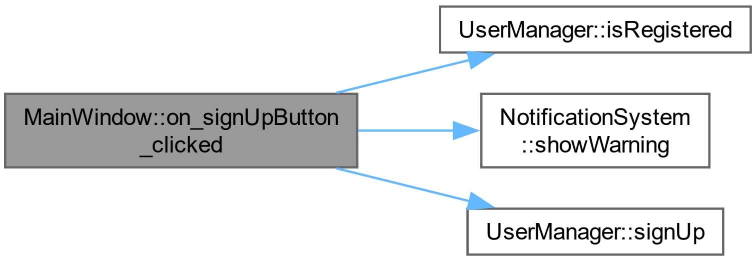
- Name and surname must be at least 3 characters long.
- Email must contain ".com" to be considered valid.
- Password must be at least 3 characters long.
- The email must not already be registered.

If all validations pass, the user is registered through the `userManager` and navigated back to the login page. If any validation fails, an appropriate warning is displayed to the user.

Note

This method uses the `notificationSystem` to show warnings or errors.

Here is the call graph for this function:


UserManager::isRegistered
MainWindow::on_signUpButton _clicked
NotificationSystem ::showWarning
UserManager::signUp

6.4.3.27 on_sizeComboBox_currentTextChanged

```
void MainWindow::on_sizeComboBox_currentTextChanged (
             const QString & arg1)  [private], [slot]
```

Updates the selected size of a product.

Handles the text change event for the size combo box.

Parameters

arg1	The selected size as a QString.

This method maps the selected size string from the combo box to the corresponding `Product::SIZE` enum value and updates the selected size of the currently active product.

Parameters

arg1	The selected size as a string.

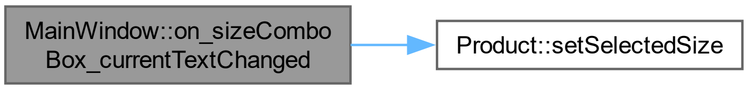
Note

This method assumes that the combo box contains valid size options matching the `Product::SIZE` enum values. If no product is currently selected, the size change is ignored.

See also

Product::SIZE, Product::setSelectedSize

Here is the call graph for this function:



6.4.3.28 on_submitCommentButtonClicked

```
void MainWindow::on_submitCommentButtonClicked ()  [private], [slot]
```

Submits a comment for the selected product.

Handles the click event for the "submit comment" button.

Validates and adds the user's comment to the product's comment list.

This method allows the current user to submit a comment for the selected product. The following conditions are validated before submitting the comment:

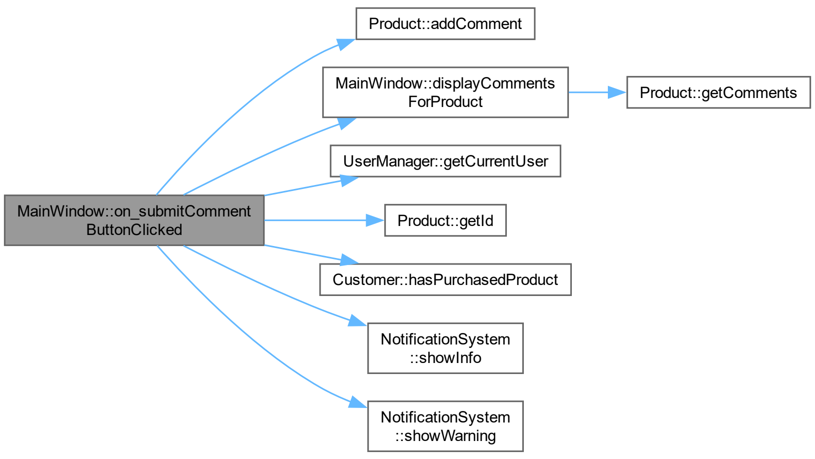
- The user must have purchased the product.
- The comment text must not be empty.

If the comment is valid, it is added to the product, and the comments section is updated. Informational or warning notifications are displayed based on the action or validation results.

Note

The comment input field is cleared after a successful submission.

Here is the call graph for this function:


MainWindow::on_submitCommentButtonClicked
Product::addComment
MainWindow::displayCommentsForProduct
Product::getComments
UserManager::getCurrentUser
Product::getId
Customer::hasPurchasedProduct
NotificationSystem::showInfo
NotificationSystem::showWarning

6.4.3.29 setProductButton

```
void MainWindow::setProductButton (
             QPushButton * button,
             const QString & picturePath)  [private], [slot]
```

Sets the product button with the given picture.

Configures a QPushButton to represent a product with an image.

Configures the button's icon and style.

Parameters

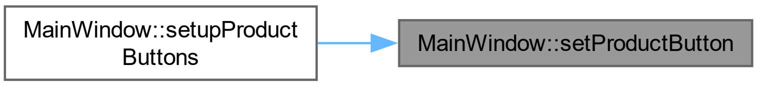
button	The QPushButton to configure.
picturePath	The path to the product's picture.

This method sets the icon of the given button to the specified image, removes the button's background for a transparent look, and ensures the button is displayed in a flat style.

Parameters

button	A pointer to the QPushButton to be configured.
picturePath	The file path to the image to be used as the button's icon.

Here is the caller graph for this function:


MainWindow::setupProduct Buttons
MainWindow::setProductButton

6.4.3.30 setupProductButtons

```
void MainWindow::setupProductButtons ()  [private], [slot]
```

Initializes product buttons.

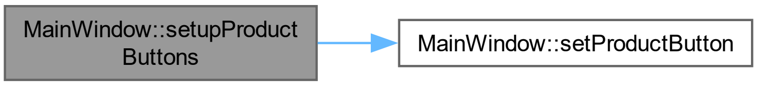
Initializes and configures product buttons for the UI.

This method sets up product buttons for clothes and electronics categories. Each button is configured with the corresponding product's image and mapped to the respective product object for interaction. Buttons without corresponding products are hidden to maintain a clean UI.

Note

The product images are loaded from the `products` vector, and the buttons are categorized as clothes and electronics.

Here is the call graph for this function:


MainWindow::setupProduct Buttons
MainWindow::setProductButton

6.4.3.31 showDiscountColors

`void MainWindow::showDiscountColors () [private], [slot]`

Displays discount color codes on the GUI.

Displays the discount colors and labels on the UI.

This method sets the background color, text color, and label text for each discount level. The discounts are displayed as visually distinct labels with different colors representing varying percentages of discounts.

- Label 1: 10% Off (Red)
- Label 2: 20% Off (Olive Green)
- Label 3: 30% Off (Steel Blue)
- Label 4: 50% Off (Gold)

6.4.3.32 spinWheel

`void MainWindow::spinWheel () [private], [slot]`

Spins the discount wheel.

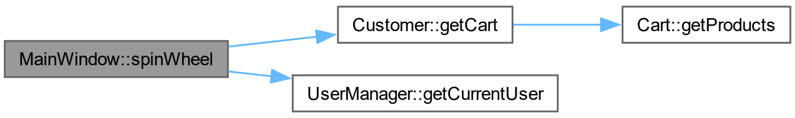
Initiates the spinning of the discount wheel.

This method checks if the user's cart is empty before allowing the wheel to spin. If the cart is empty, it displays a warning message and prevents the spin action. Otherwise, it resets the wheel's position, sets the initial speed, and starts the spinning animation.

Note

The spinning animation is controlled by a timer that updates the wheel's rotation.

Here is the call graph for this function:



6.4.4 Member Data Documentation

6.4.4.1 blinkCount

```
int MainWindow::blinkCount  [private]
```

6.4.4.2 blinkState

```
bool MainWindow::blinkState  [private]
```

6.4.4.3 blinkTimer

```
QTimer* MainWindow::blinkTimer  [private]
```

6.4.4.4 buttonProductMap

```
QMap<QPushButton *, Product *> MainWindow::buttonProductMap  [private]
```

Maps buttons to corresponding products.

6.4.4.5 chosenSliceIndex

```
int MainWindow::chosenSliceIndex  [private]
```

6.4.4.6 colorMap

```
QMap<int, QString> MainWindow::colorMap  [private]
```

6.4.4.7 commentListWidget

```
QListWidget* MainWindow::commentListWidget  [private]
```

Widget for displaying comments.

6.4.4.8 currentAngle

```
int MainWindow::currentAngle  [private]
```

6.4.4.9 currentDiscount

```
int MainWindow::currentDiscount  [private]
```

6.4.4.10 currentProduct

```
Product* MainWindow::currentProduct  [private]
```

The currently selected product.

6.4.4.11 currentSpeed

```
double MainWindow::currentSpeed  [private]
```

6.4.4.12 deceleration

```
double MainWindow::deceleration  [private]
```

6.4.4.13 discountApplied

```
bool MainWindow::discountApplied  [private]
```

6.4.4.14 discountMap

```
QMap<QString, int> MainWindow::discountMap  [private]
```

6.4.4.15 giftWrapFee

```
const double MainWindow::giftWrapFee = 10.0  [private]
```

Constants.

Fee for gift wrapping.

6.4.4.16 minSpeed

```
double MainWindow::minSpeed  [private]
```

6.4.4.17 notificationSystem

```
NotificationSystem* MainWindow::notificationSystem  [private]
```

Handles user notifications.

6.4.4.18 productButtons

```
QVector<QPushButton *> MainWindow::productButtons  [private]
```

Buttons for product selection.

6.4.4.19 products

```
QVector<Product> MainWindow::products  [private]
```

List of products displayed in the application.

6.4.4.20 scene

```
QGraphicsScene* MainWindow::scene  [private]
```

Graphics and discount wheel elements.

6.4.4.21 slices

```
QList<QGraphicsPathItem*> MainWindow::slices  [private]
```

6.4.4.22 targetAngle

```
int MainWindow::targetAngle  [private]
```

6.4.4.23 timer

```
QTimer* MainWindow::timer  [private]
```

6.4.4.24 ui

```
Ui::MainWindow* MainWindow::ui  [private]
```

The UI object for the MainWindow.

6.4.4.25 wheel

```
QGraphicsEllipseItem* MainWindow::wheel  [private]
```

6.4.4.26 wheelRewards

```
QStringList MainWindow::wheelRewards  [private]
```

The documentation for this class was generated from the following files:

- mainwindow.h
- mainwindow.cpp

6.5 NotificationSystem Class Reference

The NotificationSystem class provides an interface for displaying informational, warning, and error messages in a GUI application.

```
#include <notificationsystem.h>
```

Public Member Functions

- NotificationSystem (QWidget *parent=nullptr)

  Constructs a NotificationSystem object.
- void showInfo (const QString &title, const QString &message)

  Displays an informational message box.
- void showWarning (const QString &title, const QString &message)

  Displays a warning message box.
- void showError (const QString &title, const QString &message)

  Displays an error message box.

Private Attributes

- QWidget * parentWidget

  The parent widget for displaying message boxes.

6.5.1 Detailed Description

The NotificationSystem class provides an interface for displaying informational, warning, and error messages in a GUI application.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 NotificationSystem()

```
NotificationSystem::NotificationSystem (
             QWidget * parent = nullptr)  [explicit]
```

Constructs a NotificationSystem object.

Parameters

parent	The parent widget for displaying message boxes. Defaults to nullptr.

Initializes the NotificationSystem with a parent widget for displaying message boxes.

Parameters

parent	The parent widget. Defaults to nullptr.

6.5.3 Member Function Documentation

6.5.3.1 showError()

```
void NotificationSystem::showError (
             const QString & title,
             const QString & message)
```

Displays an error message box.

Parameters

title	The title of the message box.
message	The error message to display.

This method uses QMessageBox to display an error dialog with the specified title and message.

Parameters

title	The title of the message box.
message	The error message to display.

6.5.3.2 showInfo()

```
void NotificationSystem::showInfo (
             const QString & title,
             const QString & message)
```

Displays an informational message box.

Parameters

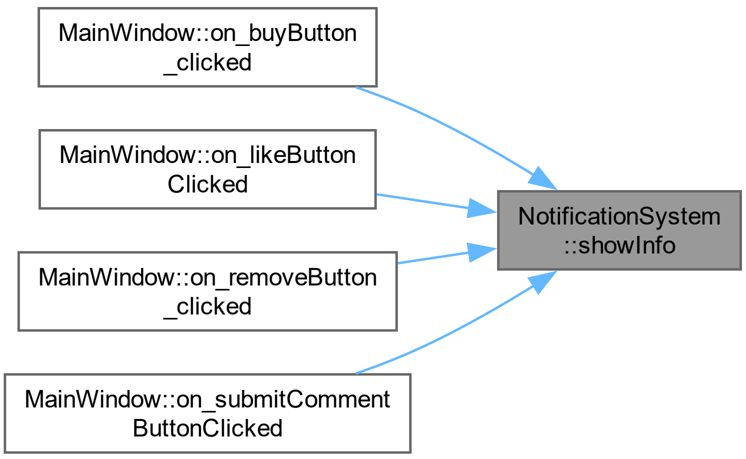
title	The title of the message box.
message	The informational message to display.

This method uses QMessageBox to display an information dialog with the specified title and message.

Parameters

title	The title of the message box.
message	The informational message to display.

Here is the caller graph for this function:


MainWindow::on_buyButton_clicked
MainWindow::on_likeButtonClicked
MainWindow::on_removeButton_clicked
MainWindow::on_submitCommentButtonClicked
NotificationSystem::showInfo

6.5.3.3 showWarning()

```
void NotificationSystem::showWarning (
             const QString & title,
             const QString & message)
```

Displays a warning message box.

Parameters

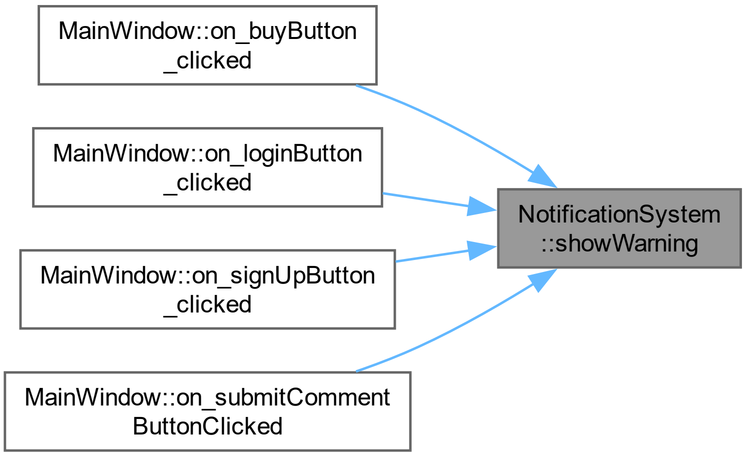
title	The title of the message box.
message	The warning message to display.

This method uses QMessageBox to display a warning dialog with the specified title and message.

Parameters

title	The title of the message box.
message	The warning message to display.

Here is the caller graph for this function:


MainWindow::on_buyButton_clicked
MainWindow::on_loginButton_clicked
MainWindow::on_signUpButton_clicked
MainWindow::on_submitCommentButtonClicked
NotificationSystem::showWarning

6.5.4 Member Data Documentation

6.5.4.1 parentWidget

```
QWidget* NotificationSystem::parentWidget  [private]
```

The parent widget for displaying message boxes.

The documentation for this class was generated from the following files:

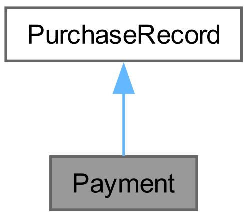
- notificationsystem.h
- notificationsystem.cpp

6.6 Payment Class Reference

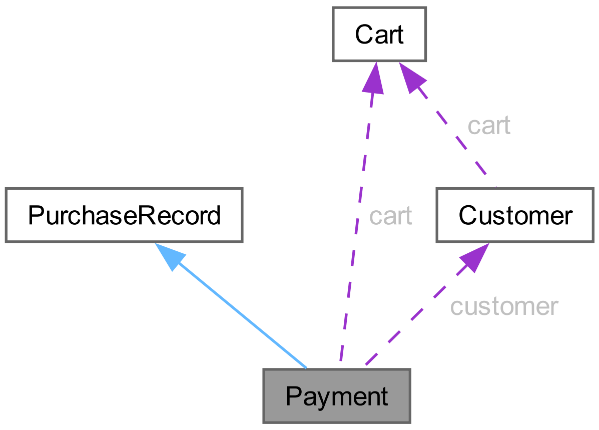
Represents a payment process, including discounts and customer information.

```
#include <payment.h>
```

Inheritance diagram for Payment:



Collaboration diagram for Payment:



Public Member Functions

- Payment (Customer customer, Cart cart, Discount discount)

  Constructs a Payment object.
- void setDiscount (Discount discount)

  Sets the discount type.
- Discount getDiscount () const

  Retrieves the currently applied discount.

- void setCustomer (const Customer &customer)

  Sets the customer associated with the payment.
- Customer getCustomer () const

  Retrieves the customer associated with the payment.
- void setCart (const Cart &cart)

  Sets the cart associated with the payment.
- Cart getCart () const

  Retrieves the cart associated with the payment.
- double grandTotal () const

  Calculates the grand total after applying discounts.
- double total () const

  Calculates the total cost of all items in the cart.
- double applyDiscount () const

  Calculates the discount amount.
- std::string discountPercentage () const

  Converts the discount type to a string representation.
- void applyDiscountCode (const QString &code)

  Applies a discount code to the payment.
- QString getRecordDetails () const override

  Retrieves the details of the payment record.

Public Member Functions inherited from PurchaseRecord

- PurchaseRecord ()

  Default constructor for the PurchaseRecord class.
- PurchaseRecord (QDateTime date, double amount)

  Constructs a PurchaseRecord with the specified date and amount.
- QDateTime getPurchaseDate () const

  Retrieves the date and time of the purchase.
- void setPurchaseDate (const QDateTime &date)

  Sets the date and time of the purchase.
- double getTotalAmount () const

  Retrieves the total amount of the purchase.
- void setTotalAmount (double amount)

  Sets the total amount of the purchase.
- virtual ∼PurchaseRecord ()=default

  Virtual destructor for the PurchaseRecord class.

Private Attributes

- Customer customer

  The customer making the payment.
- Cart cart

  The cart associated with the payment.
- Discount discount

  The discount applied to the payment.

Additional Inherited Members

Protected Attributes inherited from PurchaseRecord

- QDateTime purchaseDate

    The date and time of the purchase.
- double totalAmount

    The total amount of the purchase.

6.6.1 Detailed Description

Represents a payment process, including discounts and customer information.

The Payment class extends the PurchaseRecord class and provides functionality for calculating totals, applying discounts, and retrieving payment details.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 Payment()

```
Payment::Payment (
            Customer customer,
            Cart cart,
            Discount discount)
```

Constructs a Payment object.

Constructs a Payment object with the given customer, cart, and discount.

Parameters

customer	The customer making the payment.
cart	The shopping cart being purchased.
discount	The discount type applied to the payment.

6.6.3 Member Function Documentation

6.6.3.1 applyDiscount()

```
double Payment::applyDiscount () const
```

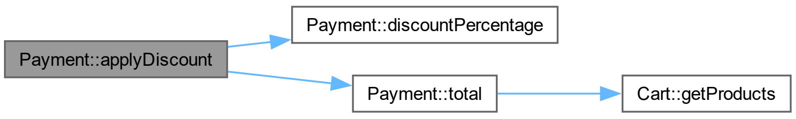
Calculates the discount amount.

Calculates the discount amount based on the applied discount type.

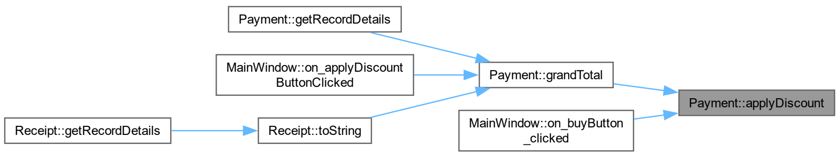
Returns

The total discount applied.

Here is the call graph for this function:



Here is the caller graph for this function:


Payment::getRecordDetails
MainWindow::on_applyDiscount ButtonClicked
Payment::grandTotal
Payment::applyDiscount
Receipt::getRecordDetails
Receipt::toString
MainWindow::on_buyButton _clicked

6.6.3.2 applyDiscountCode()

```
void Payment::applyDiscountCode (
             const QString & code)
```

Applies a discount code to the payment.

Parameters

code	The discount code as a QString.

6.6.3.3 discountPercentage()

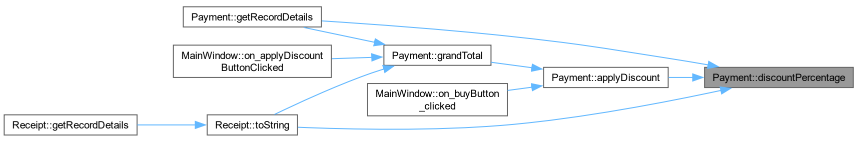
```
std::string Payment::discountPercentage () const
```

Converts the discount type to a string representation.

Returns

A string describing the discount percentage.

Here is the caller graph for this function:


Payment::getRecordDetails
MainWindow::on_applyDiscount ButtonClicked
Payment::grandTotal
Payment::applyDiscount
Payment::discountPercentage
MainWindow::on_buyButton _clicked
Receipt::getRecordDetails
Receipt::toString

6.6.3.4 getCart()

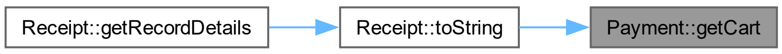
```
Cart Payment::getCart () const
```

Retrieves the cart associated with the payment.

Returns

A Cart object representing the shopping cart.

Here is the caller graph for this function:



6.6.3.5 getCustomer()

```
Customer Payment::getCustomer () const
```

Retrieves the customer associated with the payment.

Returns

A Customer object representing the customer.

6.6.3.6 getDiscount()

```
Discount Payment::getDiscount () const
```

Retrieves the currently applied discount.

Returns

The discount type.

6.6.3.7 getRecordDetails()

```
QString Payment::getRecordDetails () const  [override], [virtual]
```

Retrieves the details of the payment record.

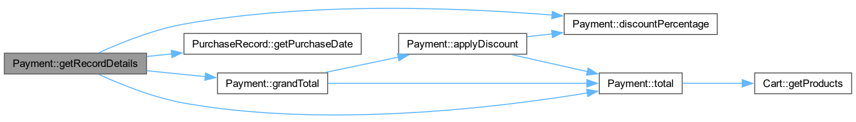
Retrieves the details of the payment record as a formatted QString.

Returns

A QString containing the payment details.

Implements PurchaseRecord.

Here is the call graph for this function:


Payment::getRecordDetails
PurchaseRecord::getPurchaseDate
Payment::grandTotal
Payment::applyDiscount
Payment::discountPercentage
Payment::total
Cart::getProducts

6.6.3.8 grandTotal()

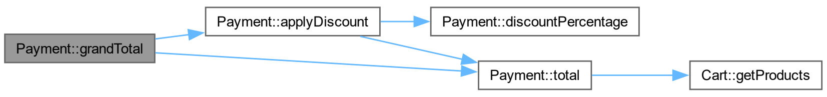
```
double Payment::grandTotal () const
```

Calculates the grand total after applying discounts.

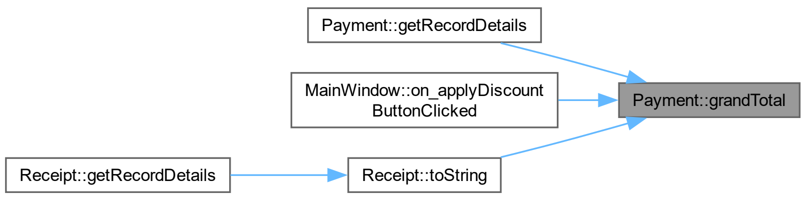
Returns

The total price after discounts.

Here is the call graph for this function:



Here is the caller graph for this function:


Payment::getRecordDetails
MainWindow::on_applyDiscount ButtonClicked
Payment::grandTotal
Receipt::getRecordDetails
Receipt::toString

6.6.3.9 setCart()

```
void Payment::setCart (
            const Cart & cart)
```

Sets the cart associated with the payment.

Parameters

cart	The cart object to set.

6.6.3.10 setCustomer()

```
void Payment::setCustomer (
            const Customer & customer)
```

Sets the customer associated with the payment.

Parameters

customer	The customer object to set.

6.6.3.11 setDiscount()

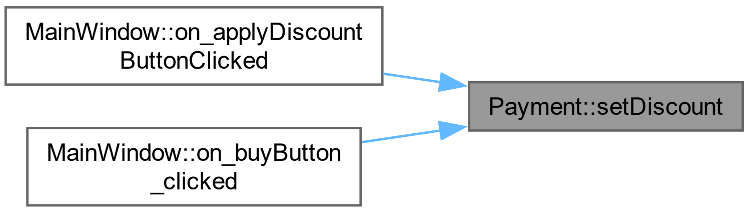
```
void Payment::setDiscount (
            Discount discount)
```

Sets the discount type.

Parameters

discount	The discount to apply.

Here is the caller graph for this function:


MainWindow::on_applyDiscount ButtonClicked
MainWindow::on_buyButton _clicked
Payment::setDiscount

6.6.3.12 total()

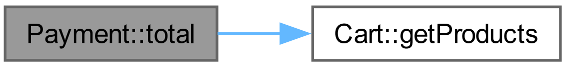
```
double Payment::total ( ) const
```

Calculates the total cost of all items in the cart.

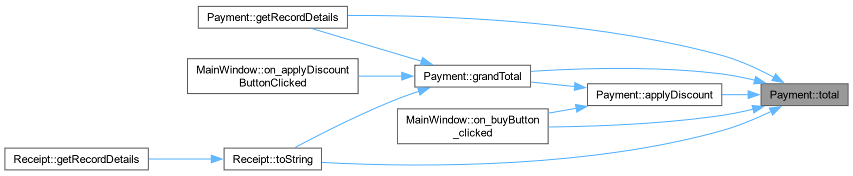
Returns

The total price without discounts.

Here is the call graph for this function:



Here is the caller graph for this function:


Payment::getRecordDetails
MainWindow::on_applyDiscount ButtonClicked
Payment::grandTotal
Payment::applyDiscount
Payment::total
MainWindow::on_buyButton _clicked
Receipt::getRecordDetails
Receipt::toString

6.6.4 Member Data Documentation

6.6.4.1 cart

```
Cart Payment::cart [private]
```

The cart associated with the payment.

6.6.4.2 customer

```
Customer Payment::customer [private]
```

The customer making the payment.

6.6.4.3 discount

```
Discount Payment::discount  [private]
```

The discount applied to the payment.

The documentation for this class was generated from the following files:

- payment.h
- payment.cpp

6.7 Product Class Reference

Represents a product in the Cartify system.

```
#include <product.h>
```

Public Types

- enum class SIZE {
  XSMALL , SMALL , MEDIUM , LARGE ,
  XLARGE }

  Enum for available product sizes.

Public Member Functions

- Product ()

  Default constructor for the Product class.
- Product (int id, QString picturePath, QString explanation, double cost, int likeCount)

  Constructs a Product with the given details.
- Product (const Product &temp)

  Copy constructor for the Product class.
- int getId () const

  Retrieves the product's ID.
- void setId (int value)

  Sets the product's ID.
- QString getPicturePath () const

  Retrieves the picture path of the product.
- void setPicturePath (const QString &value)

  Sets the picture path for the product.
- QString getExplanation () const

  Retrieves the product's description.
- void setExplanation (const QString &value)

  Sets the product's description.
- double getCost () const

  Retrieves the product's cost.
- void setCost (double value)

  Sets the product's cost.
- int getLikeCount () const

Retrieves the number of likes for the product.
- void setLikeCount (int value)

  Sets the like count for the product.
- QVector< QString > getComments () const

  Retrieves the comments associated with the product.
- void setComments (const QVector< QString > &value)

  Sets the comments for the product.
- void addComment (const QString &comment)

  Adds a comment to the product.
- void likeProduct ()

  Likes the product, incrementing the like count.
- void unlikeProduct ()

  Unlikes the product, decrementing the like count.
- SIZE getSelectedSize () const

  Retrieves the selected size of the product.
- void setSelectedSize (SIZE size)

  Sets the selected size for the product.
- QString getSizeString () const

  Retrieves the selected size as a string.
- QString toQString () const

  Converts the product details to a QString.

Public Attributes

- SIZE selectedSize

  The selected size of the product.

Private Attributes

- int id

  Unique identifier for the product.
- QString picturePath

  Path to the product's image.
- QString explanation

  Description of the product.
- double cost

  Cost of the product.
- int likeCount

  Number of likes the product has received.
- QVector< QString > comments

  Comments associated with the product.

6.7.1 Detailed Description

Represents a product in the Cartify system.

The Product class contains details about a product, including its ID, picture path, description, cost, like count, comments, and size.

6.7.2 Member Enumeration Documentation

6.7.2.1 SIZE

```
enum class Product::SIZE  [strong]
```

Enum for available product sizes.

Enumerator

XSMALL	Extra Small size.
SMALL	Small size.
MEDIUM	Medium size.
LARGE	Large size.
XLARGE	Extra Large size.

6.7.3 Constructor & Destructor Documentation

6.7.3.1 Product() [1/3]

```
Product::Product ()
```

Default constructor for the Product class.

Initializes the product with default values.

6.7.3.2 Product() [2/3]

```
Product::Product (
             int id,
             QString picturePath,
             QString explanation,
             double cost,
             int likeCount)
```

Constructs a Product with the given details.

Parameters

id	The unique identifier for the product.
picturePath	The file path to the product's image.
explanation	The description of the product.
cost	The cost of the product.
likeCount	The initial like count for the product.

6.7.3.3 Product() [3/3]

```
Product::Product (
             const Product & temp)
```

Copy constructor for the Product class.

Parameters

temp	The product to copy.

Copies the details of an existing product.

Parameters

temp	The product to copy.

6.7.4 Member Function Documentation

6.7.4.1 addComment()

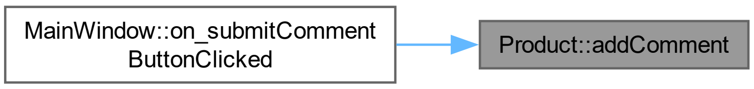
```
void Product::addComment (
             const QString & comment)
```

Adds a comment to the product.

Parameters

comment	The comment to add.

Here is the caller graph for this function:



6.7.4.2 getComments()

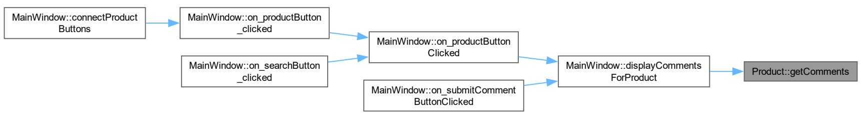
```
QVector< QString > Product::getComments () const
```

Retrieves the comments associated with the product.

Returns

A QVector of comments.

Here is the caller graph for this function:


MainWindow::connectProduct Buttons
MainWindow::on_productButton _clicked
MainWindow::on_searchButton _clicked
MainWindow::on_productButton Clicked
MainWindow::on_submitComment ButtonClicked
MainWindow::displayComments ForProduct
Product::getComments

6.7.4.3 getCost()

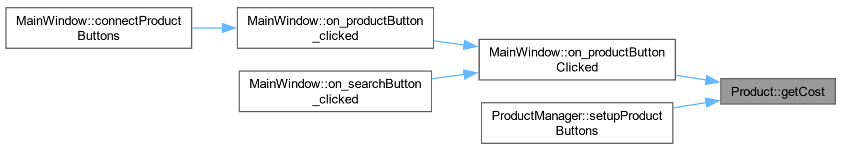
```
double Product::getCost () const
```

Retrieves the product's cost.

Returns

The cost of the product.

Here is the caller graph for this function:


MainWindow::connectProduct Buttons
MainWindow::on_productButton _clicked
MainWindow::on_searchButton _clicked
MainWindow::on_productButton Clicked
ProductManager::setupProduct Buttons
Product::getCost

6.7.4.4 getExplanation()

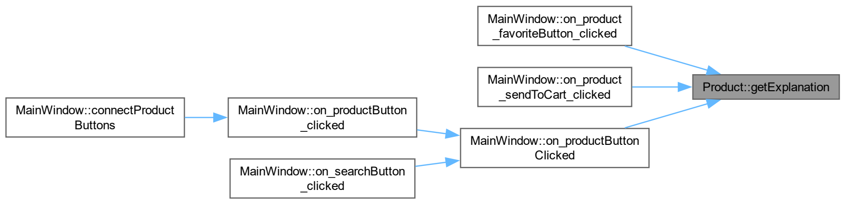
```
QString Product::getExplanation () const
```

Retrieves the product's description.

Returns

The description of the product.

Here is the caller graph for this function:


MainWindow::on_product _favoriteButton_clicked
MainWindow::on_product _sendToCart_clicked
Product::getExplanation
MainWindow::connectProduct Buttons
MainWindow::on_productButton _clicked
MainWindow::on_productButton Clicked
MainWindow::on_searchButton _clicked

6.7.4.5 getId()

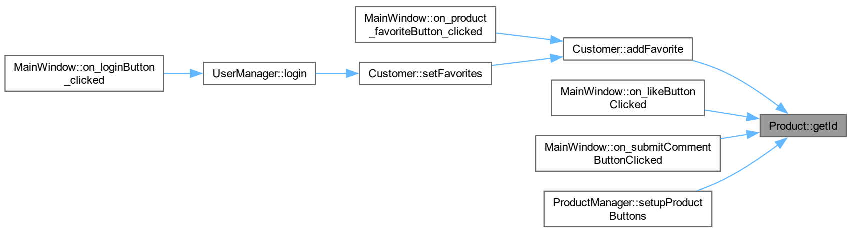
```
int Product::getId () const
```

Retrieves the product's ID.

Returns

The ID of the product.

Here is the caller graph for this function:


MainWindow::on_loginButton _clicked
UserManager::login
Customer::setFavorites
MainWindow::on_product _favoriteButton_clicked
Customer::addFavorite
MainWindow::on_likeButton Clicked
MainWindow::on_submitComment ButtonClicked
ProductManager::setupProduct Buttons
Product::getId

6.7.4.6 getLikeCount()

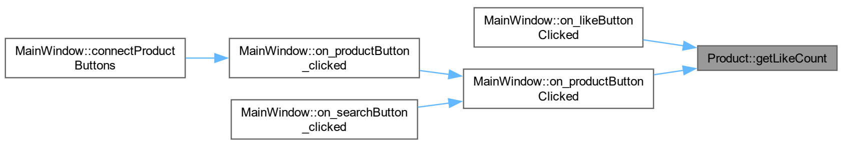
```
int Product::getLikeCount () const
```

Retrieves the number of likes for the product.

Returns

The number of likes.

Here is the caller graph for this function:


MainWindow::connectProduct Buttons
MainWindow::on_productButton _clicked
MainWindow::on_searchButton _clicked
MainWindow::on_likeButton Clicked
MainWindow::on_productButton Clicked
Product::getLikeCount

6.7.4.7 getPicturePath()

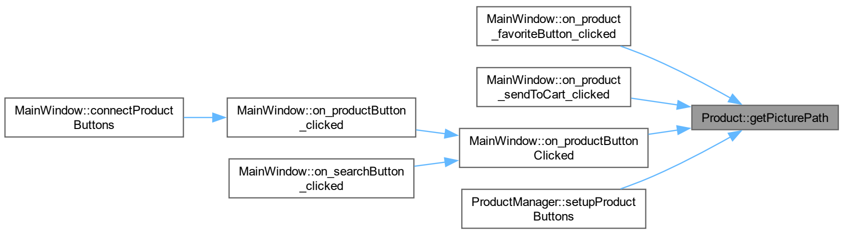
```
QString Product::getPicturePath () const
```

Retrieves the picture path of the product.

Returns

The file path to the product's image.

Here is the caller graph for this function:


MainWindow::on_product _favoriteButton_clicked
MainWindow::on_product _sendToCart_clicked
MainWindow::connectProduct Buttons
MainWindow::on_productButton _clicked
MainWindow::on_productButton Clicked
Product::getPicturePath
MainWindow::on_searchButton _clicked
ProductManager::setupProduct Buttons

6.7.4.8 getSelectedSize()

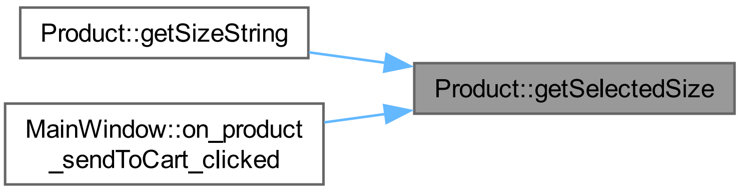
```
Product::SIZE Product::getSelectedSize () const
```

Retrieves the selected size of the product.

Returns

The selected size.

Here is the caller graph for this function:



6.7.4.9 getSizeString()

```
QString Product::getSizeString () const
```

Retrieves the selected size as a string.

Returns

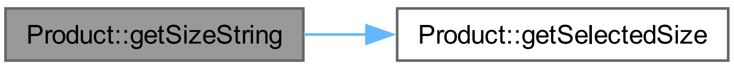
A QString representing the size (e.g., "XS", "S").

Converts the selected size enum to a human-readable string.

Returns

A QString representing the size (e.g., "XS", "S").

Here is the call graph for this function:

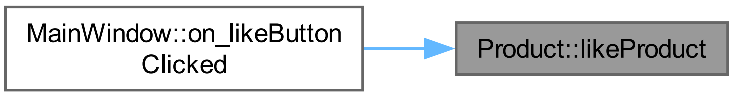


6.7.4.10 likeProduct()

```
void Product::likeProduct ()
```

Likes the product, incrementing the like count.

Here is the caller graph for this function:



6.7.4.11 setComments()

```
void Product::setComments (
            const QVector< QString > & value)
```

Sets the comments for the product.

Parameters

value	A QVector of comments to set.

6.7.4.12 setCost()

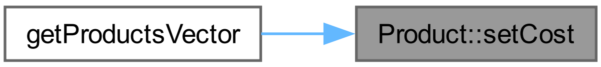
```
void Product::setCost (
             double value)
```

Sets the product's cost.

Parameters

value	The cost to set.

Here is the caller graph for this function:



6.7.4.13 setExplanation()

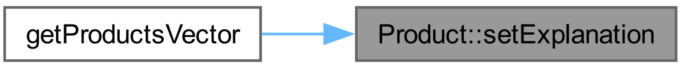
```
void Product::setExplanation (
             const QString & value)
```

Sets the product's description.

Parameters

value	The description to set.

Here is the caller graph for this function:



6.7.4.14 setId()

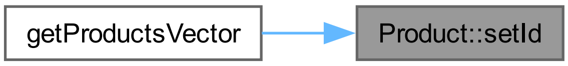
```
void Product::setId (
             int value)
```

Sets the product's ID.

Parameters

value	The ID to set.

Here is the caller graph for this function:



6.7.4.15 setLikeCount()

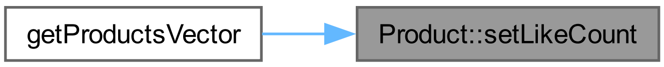
```
void Product::setLikeCount (
             int value)
```

Sets the like count for the product.

Parameters

value	The like count to set.

Here is the caller graph for this function:



6.7.4.16 setPicturePath()

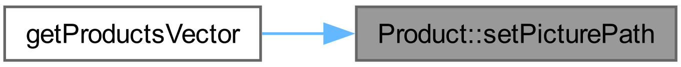
```
void Product::setPicturePath (
             const QString & value)
```

Sets the picture path for the product.

Parameters

value	The file path to set.

Here is the caller graph for this function:



6.7.4.17 setSelectedSize()

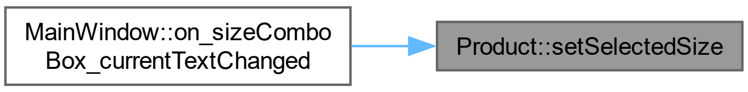
```
void Product::setSelectedSize (
             SIZE size)
```

Sets the selected size for the product.

Parameters

size	The size to set.

Here is the caller graph for this function:



6.7.4.18 toQString()

```
QString Product::toQString () const
```

Converts the product details to a QString.

Returns

A QString containing the product ID and cost.

6.7.4.19 unlikeProduct()

```
void Product::unlikeProduct ()
```

Unlikes the product, decrementing the like count.

Ensures the like count does not go below zero.

6.7.5 Member Data Documentation

6.7.5.1 comments

```
QVector<QString> Product::comments [private]
```

Comments associated with the product.

6.7.5.2 cost

```
double Product::cost [private]
```

Cost of the product.

6.7.5.3 explanation

```
QString Product::explanation [private]
```

Description of the product.

6.7.5.4 id

```
int Product::id [private]
```

Unique identifier for the product.

6.7.5.5 likeCount

```
int Product::likeCount [private]
```

Number of likes the product has received.

6.7.5.6 picturePath

```
QString Product::picturePath [private]
```

Path to the product's image.

6.7.5.7 selectedSize

```
SIZE Product::selectedSize
```

The selected size of the product.

The documentation for this class was generated from the following files:

- product.h
- product.cpp

6.8 ProductManager Class Reference

Manages products and their interactions within the Cartify system.

```
#include <productmanager.h>
```

Public Member Functions

- ProductManager (const QVector< Product > &products)

  Constructs a ProductManager object with a list of products.
- void setupProductButtons (const QVector< QPushButton * > &buttons, int startIndex=0)

  Sets up product buttons for the GUI.
- Product * getProductByIndex (int index)

  Retrieves a product by its index.

Private Attributes

- QVector< Product > products

  The list of products managed by this class.

6.8.1 Detailed Description

Manages products and their interactions within the Cartify system.

The ProductManager class handles product data and provides functionalities such as setting up product buttons for the GUI and retrieving product details.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 ProductManager()

```
ProductManager::ProductManager (
                const QVector< Product > & products)
```

Constructs a ProductManager object with a list of products.

Constructs a ProductManager object with the given list of products.

Parameters

products	A QVector containing the products to manage.

Initializes the ProductManager with the provided QVector of Product objects.

Parameters

products	A QVector containing the products to manage.

6.8.3 Member Function Documentation

6.8.3.1 getProductByIndex()

```
Product * ProductManager::getProductByIndex (
            int index)
```

Retrieves a product by its index.

Returns a pointer to the product at the specified index in the product list. Returns nullptr if the index is out of bounds.

Parameters

index	The index of the product to retrieve.

Returns

A pointer to the Product object, or nullptr if the index is invalid.

This function returns a pointer to the Product object at the specified index in the product list. If the index is out of bounds, it returns nullptr.

Parameters

index	The index of the product to retrieve.

Returns

A pointer to the Product object, or nullptr if the index is invalid.

6.8.3.2 setupProductButtons()

```
void ProductManager::setupProductButtons (
            const QVector< QPushButton * > & buttons,
            int startIndex = 0)
```

Sets up product buttons for the GUI.

Assigns icons and styles to a list of QPushButtons based on the products starting from the specified index.

Parameters

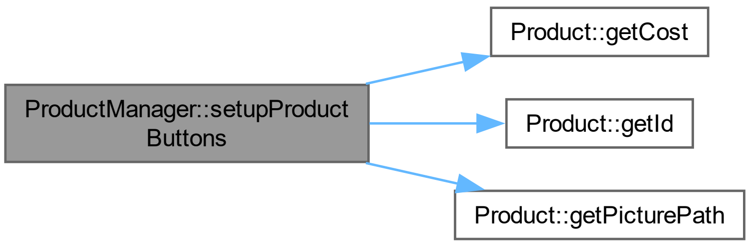
buttons	A QVector of QPushButton pointers to configure.
startIndex	The starting index in the product list. Defaults to 0.

This function assigns icons and styles to a QVector of QPushButton pointers based on the products managed by the ProductManager. The setup starts from the given index in the product list.

Parameters

buttons	A QVector of QPushButton pointers to configure.
startIndex	The starting index in the product list. Defaults to 0.

Here is the call graph for this function:



6.8.4 Member Data Documentation

6.8.4.1 products

```
QVector<Product> ProductManager::products  [private]
```

The list of products managed by this class.

The documentation for this class was generated from the following files:

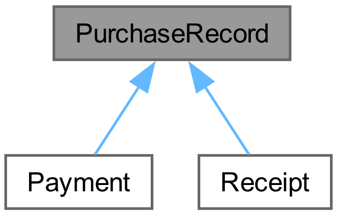
- productmanager.h
- productmanager.cpp

6.9 PurchaseRecord Class Reference

Represents a generic purchase record in the Cartify system.

```
#include <purchaserecord.h>
```

Inheritance diagram for PurchaseRecord:



Public Member Functions

- PurchaseRecord ()

  Default constructor for the PurchaseRecord class.
- PurchaseRecord (QDateTime date, double amount)

  Constructs a PurchaseRecord with the specified date and amount.
- virtual QString getRecordDetails () const =0

  Retrieves the details of the purchase record.
- QDateTime getPurchaseDate () const

  Retrieves the date and time of the purchase.
- void setPurchaseDate (const QDateTime &date)

  Sets the date and time of the purchase.
- double getTotalAmount () const

  Retrieves the total amount of the purchase.
- void setTotalAmount (double amount)

  Sets the total amount of the purchase.
- virtual ∼PurchaseRecord ()=default

  Virtual destructor for the PurchaseRecord class.

Protected Attributes

- QDateTime purchaseDate

  The date and time of the purchase.
- double totalAmount

  The total amount of the purchase.

6.9.1 Detailed Description

Represents a generic purchase record in the Cartify system.

The PurchaseRecord class provides a base for storing purchase details, including the date and total amount. It serves as an abstract base class for more specific types of purchase records.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 PurchaseRecord() [1/2]

```
PurchaseRecord::PurchaseRecord ()
```

Default constructor for the PurchaseRecord class.

Initializes the purchase date to the current date and time and the total amount to 0.

6.9.2.2 PurchaseRecord() [2/2]

```
PurchaseRecord::PurchaseRecord (
            QDateTime date,
            double amount)
```

Constructs a PurchaseRecord with the specified date and amount.

Parameters

date	The date and time of the purchase.
amount	The total amount of the purchase.

6.9.2.3 ∼PurchaseRecord()

```
virtual PurchaseRecord::∼PurchaseRecord ()  [virtual], [default]
```

Virtual destructor for the PurchaseRecord class.

Ensures proper cleanup of derived classes.

6.9.3 Member Function Documentation

6.9.3.1 getPurchaseDate()

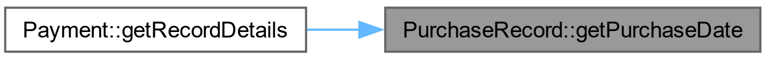
```
QDateTime PurchaseRecord::getPurchaseDate () const
```

Retrieves the date and time of the purchase.

Returns

The purchase date as a QDateTime object.

Here is the caller graph for this function:



6.9.3.2 getRecordDetails()

```
virtual QString PurchaseRecord::getRecordDetails () const  [pure virtual]
```

Retrieves the details of the purchase record.

This is a pure virtual function to be implemented by derived classes.

Returns

A QString containing the details of the purchase record.

Implemented in Payment, and Receipt.

6.9.3.3 getTotalAmount()

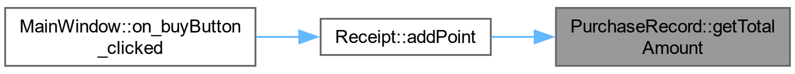
```
double PurchaseRecord::getTotalAmount () const
```

Retrieves the total amount of the purchase.

Returns

The total amount of the purchase as a double.

Here is the caller graph for this function:


MainWindow::on_buyButton_clicked
Receipt::addPoint
PurchaseRecord::getTotalAmount

6.9.3.4 setPurchaseDate()

```
void PurchaseRecord::setPurchaseDate (
             const QDateTime & date)
```

Sets the date and time of the purchase.

Parameters

date	The date and time to set.

6.9.3.5 setTotalAmount()

```
void PurchaseRecord::setTotalAmount (
             double amount)
```

Sets the total amount of the purchase.

Parameters

amount	The amount to set.

6.9.4 Member Data Documentation

6.9.4.1 purchaseDate

```
QDateTime PurchaseRecord::purchaseDate  [protected]
```

The date and time of the purchase.

6.9.4.2 totalAmount

```
double PurchaseRecord::totalAmount  [protected]
```

The total amount of the purchase.

The documentation for this class was generated from the following files:

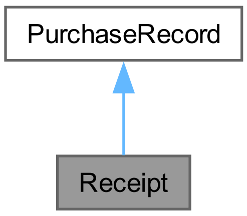
- purchaserecord.h
- purchaserecord.cpp

6.10 Receipt Class Reference

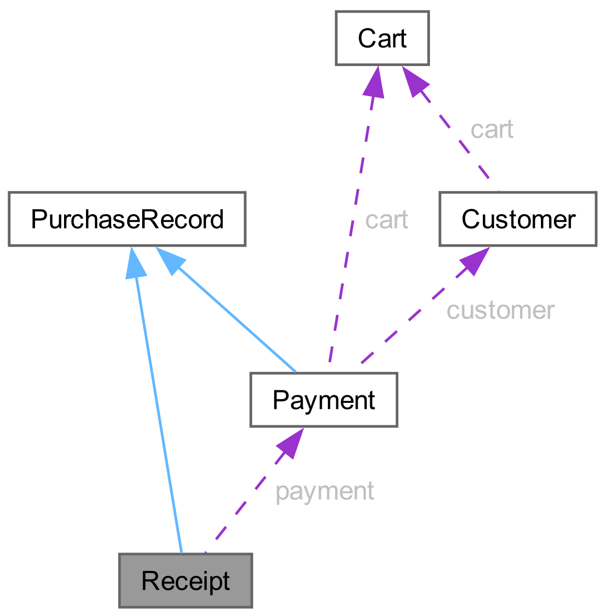
Represents a receipt for a completed purchase in the Cartify system.

```
#include <receipt.h>
```

Inheritance diagram for Receipt:



Collaboration diagram for Receipt:



Public Member Functions

- Receipt (const Payment &payment)

  Constructs a Receipt object with the given payment details.
- int addPoint ()

  Calculates the loyalty points earned from the purchase.
- QString orderNo () const

  Generates a unique order number for the receipt.
- QString toString () const

  Converts the receipt details to a formatted string.
- QString getRecordDetails () const override

  Retrieves the complete details of the receipt.

Public Member Functions inherited from PurchaseRecord

- PurchaseRecord ()

  Default constructor for the PurchaseRecord class.
- PurchaseRecord (QDateTime date, double amount)

  Constructs a PurchaseRecord with the specified date and amount.
- QDateTime getPurchaseDate () const

  Retrieves the date and time of the purchase.
- void setPurchaseDate (const QDateTime &date)

  Sets the date and time of the purchase.
- double getTotalAmount () const

Retrieves the total amount of the purchase.

- void setTotalAmount (double amount)

  Sets the total amount of the purchase.
- virtual ∼PurchaseRecord ()=default

  Virtual destructor for the PurchaseRecord class.

Private Attributes

- Payment payment

  The payment details associated with this receipt.

Additional Inherited Members

Protected Attributes inherited from PurchaseRecord

- QDateTime purchaseDate

  The date and time of the purchase.
- double totalAmount

  The total amount of the purchase.

6.10.1 Detailed Description

Represents a receipt for a completed purchase in the Cartify system.

The Receipt class extends the PurchaseRecord class and provides details about the purchased items, discounts, and total cost. It also generates a unique order number.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 Receipt()

```
Receipt::Receipt (
                 const Payment & payment)
```

Constructs a Receipt object with the given payment details.

Parameters

payment	The Payment object containing purchase details.

Initializes the receipt with payment information and sets the purchase date and total amount based on the payment details.

Parameters

payment	The Payment object containing purchase details.

6.10.3 Member Function Documentation

6.10.3.1 addPoint()

```
int Receipt::addPoint ()
```

Calculates the loyalty points earned from the purchase.

The points are calculated based on the total amount of the purchase.

Returns

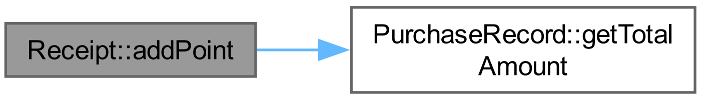
The number of points earned.

Points are calculated as 10% of the total purchase amount.

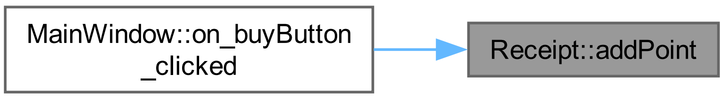
Returns

The number of points earned.

Here is the call graph for this function:



Here is the caller graph for this function:



6.10.3.2 getRecordDetails()

```
QString Receipt::getRecordDetails () const  [override], [virtual]
```

Retrieves the complete details of the receipt.

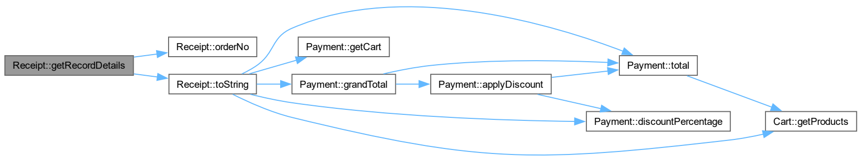
Combines the order number and receipt information into a single string.

Returns

A QString containing the full receipt details.

Implements PurchaseRecord.

Here is the call graph for this function:



6.10.3.3 orderNo()

```
QString Receipt::orderNo () const
```

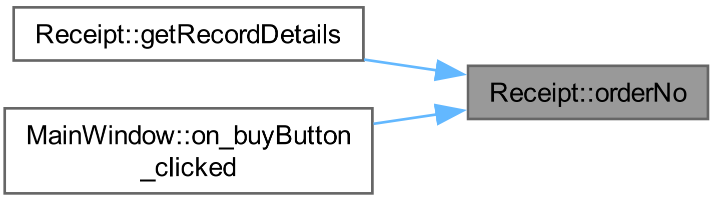
Generates a unique order number for the receipt.

The order number is a randomly generated 6-digit number.

Returns

The generated order number as a QString.

Here is the caller graph for this function:



6.10.3.4 toString()

```
QString Receipt::toString () const
```

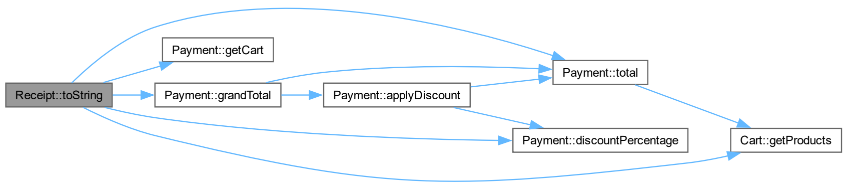
Converts the receipt details to a formatted string.

The string includes information about each purchased product, the total cost, discounts applied, and the grand total.

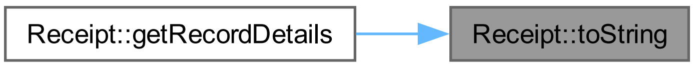
Returns

A QString containing the receipt details.

Here is the call graph for this function:



Here is the caller graph for this function:



6.10.4 Member Data Documentation

6.10.4.1 payment

```
Payment Receipt::payment [private]
```

The payment details associated with this receipt.

The documentation for this class was generated from the following files:

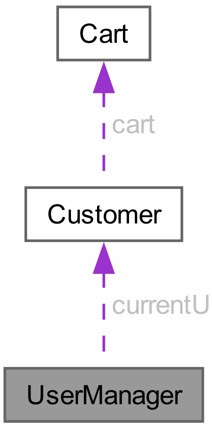
- receipt.h
- receipt.cpp

6.11 UserManager Class Reference

Manages user accounts and authentication in the Cartify system.

```
#include <usermanager.h>
```

Collaboration diagram for UserManager:



Public Member Functions

- UserManager ()

  Constructs a UserManager object with a default user.
- Customer & getCurrentUser ()

  Retrieves the current logged-in user.
- bool login (const QString &email, const QString &password, const QVector< Product > &products)

  Authenticates a user based on email and password.
- bool isRegistered (const QString &email)

  Checks if a user is already registered based on email.
- void signUp (const QString &name, const QString &surname, const QString &email, const QString &password)

  Registers a new user and stores their data.

Private Attributes

- Customer currentUser

  The currently logged-in user.

6.11.1 Detailed Description

Manages user accounts and authentication in the Cartify system.

The UserManager class handles user login, registration, and retrieval of the current user's data.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 UserManager()

```
UserManager::UserManager ()
```

Constructs a UserManager object with a default user.

Constructs a UserManager object with default values.

6.11.3 Member Function Documentation

6.11.3.1 getCurrentUser()

```
Customer & UserManager::getCurrentUser ()
```

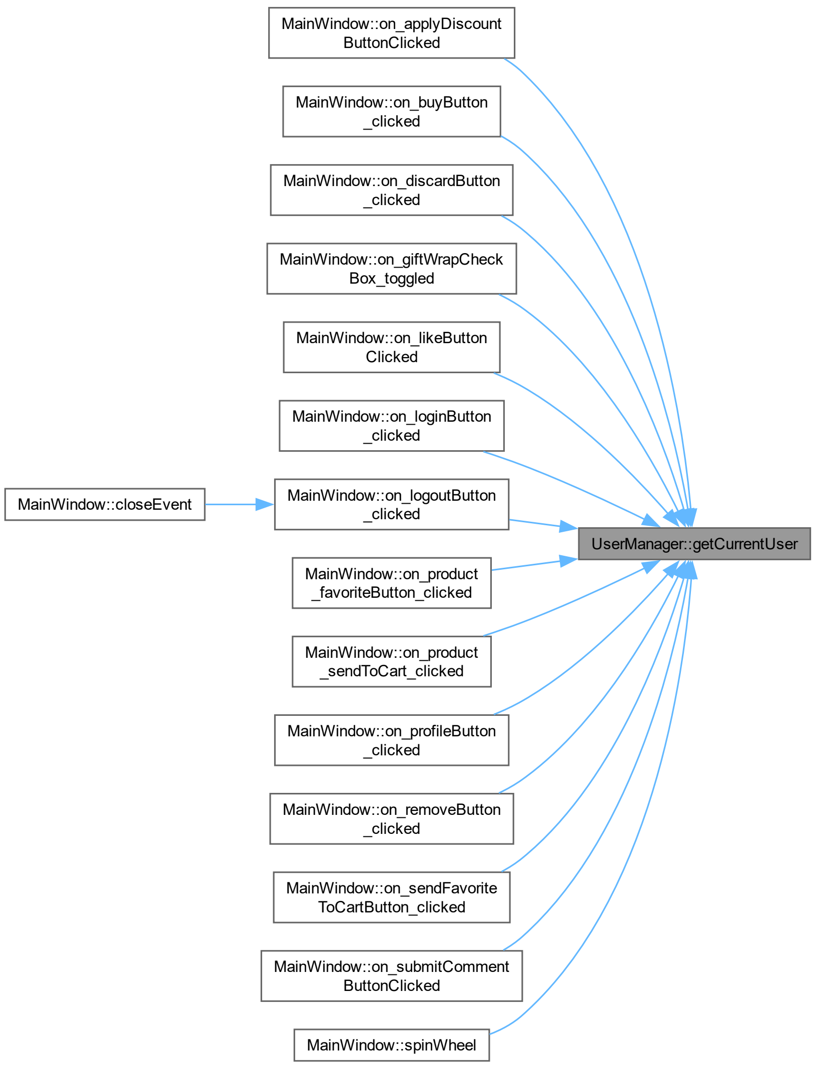
Retrieves the current logged-in user.

Retrieves the currently logged-in user.

Returns

A reference to the Customer object representing the current user.

Here is the caller graph for this function:


MainWindow::on_applyDiscount ButtonClicked
MainWindow::on_buyButton _clicked
MainWindow::on_discardButton _clicked
MainWindow::on_giftWrapCheck Box_toggled
MainWindow::on_likeButton Clicked
MainWindow::on_loginButton _clicked
MainWindow::closeEvent
MainWindow::on_logoutButton _clicked
UserManager::getCurrentUser
MainWindow::on_product _favoriteButton_clicked
MainWindow::on_product _sendToCart_clicked
MainWindow::on_profileButton _clicked
MainWindow::on_removeButton _clicked
MainWindow::on_sendFavorite ToCartButton_clicked
MainWindow::on_submitComment ButtonClicked
MainWindow::spinWheel

6.11.3.2 isRegistered()

```
bool UserManager::isRegistered (
             const QString & email)
```

Checks if a user is already registered based on email.

Checks if a user is already registered based on their email.

Searches the stored user data for a matching email.

Parameters

email	The email address to check.

Returns

True if the email is already registered, false otherwise.

Searches the user storage file for a matching email.

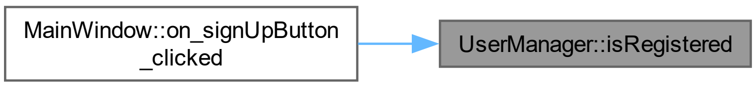
Parameters

email	The email address to check.

Returns

True if the email is already registered, false otherwise.

Here is the caller graph for this function:



6.11.3.3 login()

```
bool UserManager::login (
            const QString & email,
            const QString & password,
            const QVector< Product > & products)
```

Authenticates a user based on email and password.

Authenticates a user based on their email and password.

Searches the stored user data for a matching email and password combination. If successful, updates the current user and their favorites.

Parameters

email	The email address provided for login.
password	The password provided for login.
products	The list of products to initialize user favorites.

Returns

True if authentication is successful, false otherwise.

Searches the user storage file for a matching email and password combination. Updates the current user if authentication is successful.

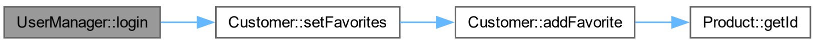
Parameters

email	The email address provided for login.
password	The password provided for login.
products	A list of products to initialize the user's favorites.

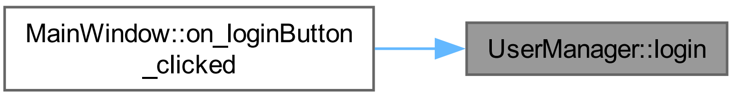
Returns

True if authentication is successful, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



6.11.3.4 signUp()

```
void UserManager::signUp (
             const QString & name,
             const QString & surname,
             const QString & email,
             const QString & password)
```

Registers a new user and stores their data.

Registers a new user by saving their details.

Saves the provided user details to the user data storage and updates the current user to the newly registered user.

Parameters

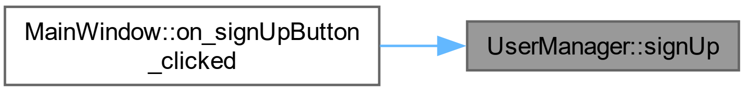
name	The first name of the user.
surname	The last name of the user.
email	The email address of the user.
password	The password for the user.

Writes the new user's details to the user storage file and updates the current user to the newly registered user.

Parameters

name	The first name of the user.
surname	The last name of the user.
email	The email address of the user.
password	The password for the user.

Here is the caller graph for this function:



6.11.4 Member Data Documentation

6.11.4.1 currentUser

```
Customer UserManager::currentUser  [private]
```

The currently logged-in user.

The documentation for this class was generated from the following files:

- usermanager.h
- usermanager.cpp

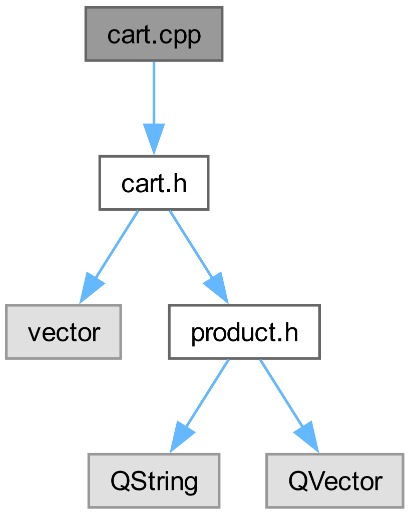
Chapter 7

File Documentation

7.1 cart.cpp File Reference

```
#include "cart.h"
```

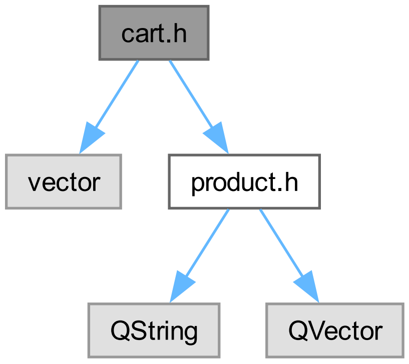
Include dependency graph for cart.cpp:



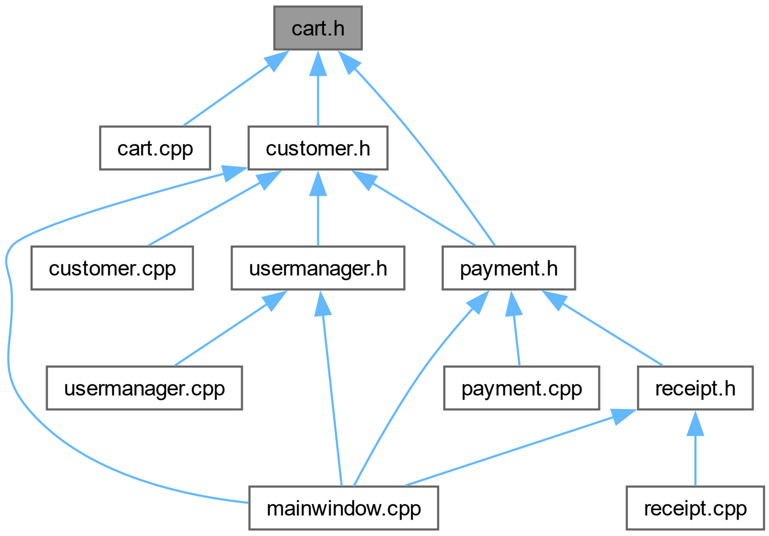
7.2 cart.h File Reference

```
#include <vector>
#include "product.h"
```

Include dependency graph for cart.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Cart

  Represents a shopping cart that holds a collection of products.

7.3 cart.h

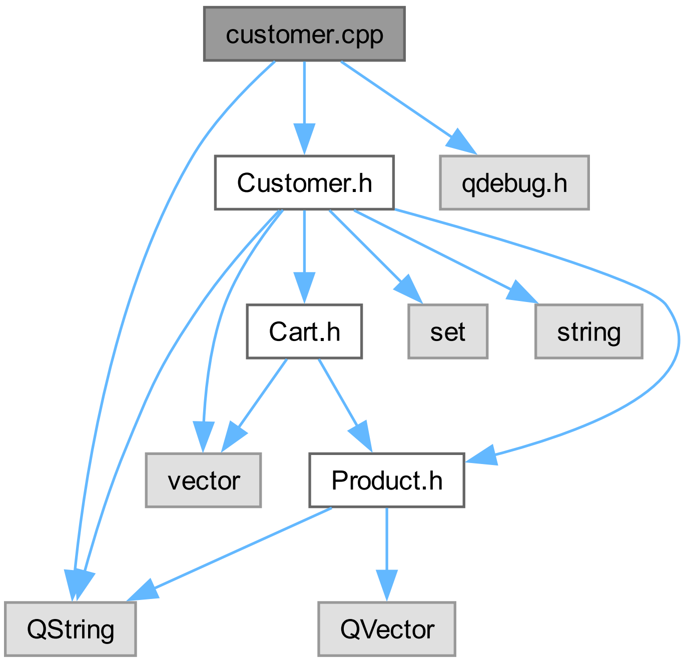
Go to the documentation of this file.

```
#ifndef CART_H
#define CART_H

#include <vector>
#include "product.h"
using namespace std;

class Cart {
private:
    vector<Product> products;

public:
    Cart();

    void addProduct(const Product& product);

    void removeProductById(int productId);

    const vector<Product>& getProducts() const;

    void clearCart();
};

#endif // CART_H
```

7.4 customer.cpp File Reference

```
#include "Customer.h"
#include <QString>
#include "qdebug.h"
```

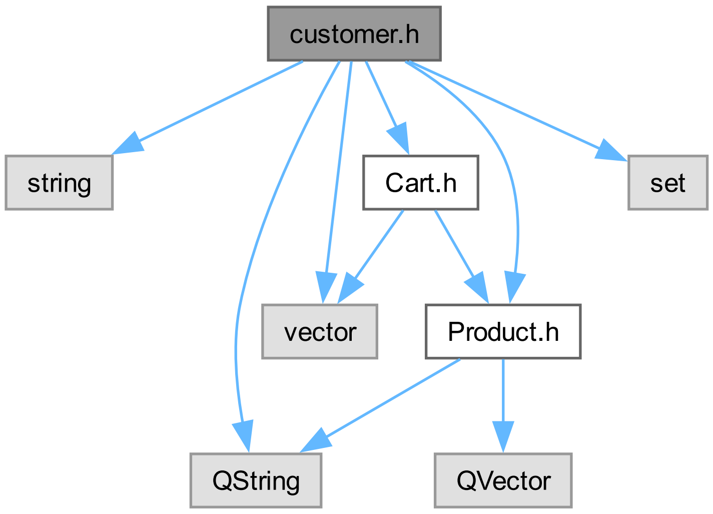
Include dependency graph for customer.cpp:



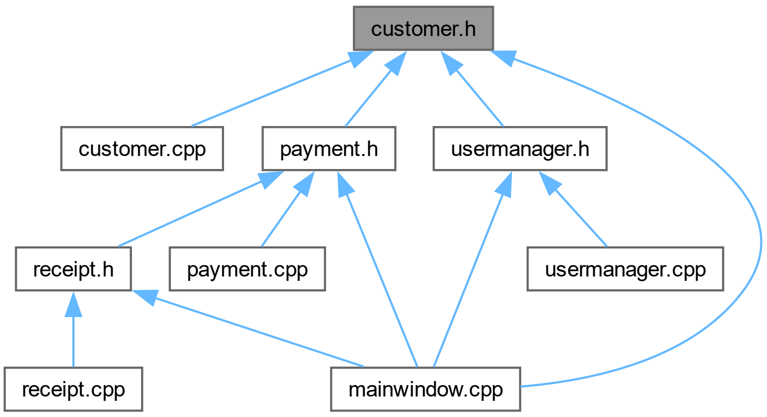
7.5 customer.h File Reference

```
#include <string>
#include <vector>
#include <QString>
#include "Product.h"
#include "Cart.h"
#include <set>
```

Include dependency graph for customer.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Customer

  Represents a customer in the Cartify system.

Enumerations

- enum class ProductType { Electronics , Clothes }

  Enum for product types.

7.5.1 Enumeration Type Documentation

7.5.1.1 ProductType

```
enum class ProductType  [strong]
```

Enum for product types.

Enumerator

Electronics	Represents electronic products.
Clothes	Represents clothing products.

7.6 customer.h

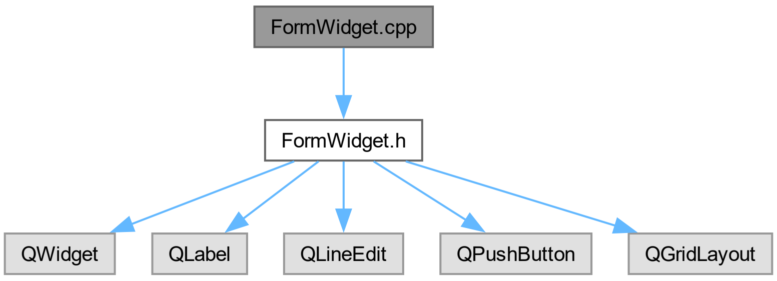
Go to the documentation of this file.

```
// Customer.h
#ifndef CUSTOMER_H
#define CUSTOMER_H

#include <string>
#include <vector>
#include <QString>
#include "Product.h"
#include "Cart.h"
#include <set>

using namespace std;

enum class ProductType {
    Electronics,
    Clothes
};

class Customer {
    std::set<int> likedProductIds;

public:


    Cart cart;
    Customer(QString name, QString surname, ProductType productType,
             QString email, QString password);

    ~Customer();


    Cart getCartObject();

    void addPoint(int amount);

    int myPoints() const;

    bool isCartEmpty(Product prod);

    bool addFavorite(Product prod);

    void removeFavorite(int id);

    QVector<Product> getFavorites();

    void setFavorites(QString favorites, QVector<Product> products);

    void addCart(Product prod);

    void removeCart(int id);

    vector<Product> getCart();

    bool likeProduct(int productId);

    bool hasLikedProduct(int productId) const;

    void addPurchasedProduct(const Product& product);

    bool hasPurchasedProduct(int productId) const;



    QString getName() const {
        return name;
    }


    void setName(const QString& newName) {
        name = newName;
    }


    QString getSurname() const {
        return surname;
    }


    void setSurname(const QString& newSurname) {
        surname = newSurname;
    }


    QString getEmail() const {
        return email;
    }


    void setPassword(const QString& newPassword) {
        password = newPassword;
    }
    //(no getter for security reasons)


    ProductType getProductType() const {
        return productType;
    }


    void setProductType(ProductType newProductType) {
        productType = newProductType;
    }


    int getPoints() const {
        return point;
    }


    void setPoints(int newPoints) {
        point = newPoints;
    }

private:
    std::vector<Product> purchasedProducts;

    QString name;
    QString surname;
    QString email;
    QString password;
    ProductType productType;
    int point;

    QVector<Product> favorites;
    QVector<Product> previousOrders;

};

#endif // CUSTOMER_H
```

7.7 FormWidget.cpp File Reference

`#include "FormWidget.h"`

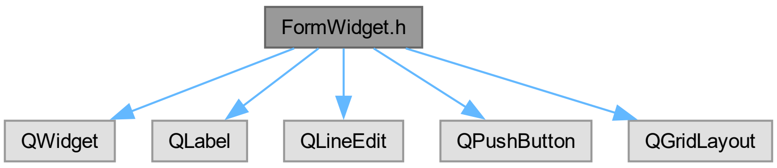
Include dependency graph for FormWidget.cpp:



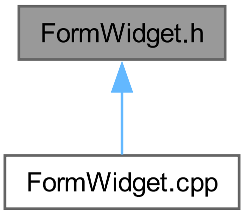
7.8 FormWidget.h File Reference

```
#include <QWidget>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QGridLayout>
```

Include dependency graph for FormWidget.h:



This graph shows which files directly or indirectly include this file:



Classes

- class FormWidget

    A widget that provides a login and sign-up form interface.

7.9 FormWidget.h

Go to the documentation of this file.

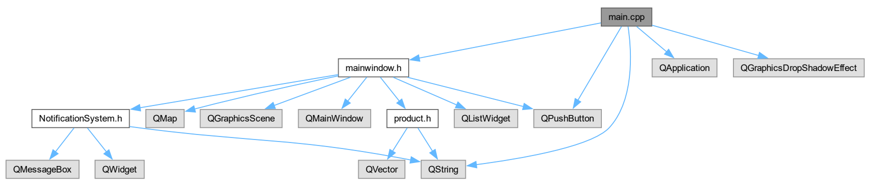
```
#ifndef FORMWIDGET_H
#define FORMWIDGET_H

#include <QWidget>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QGridLayout>

class FormWidget : public QWidget
{
    Q_OBJECT

public:
    explicit FormWidget(QWidget *parent = nullptr);

private:
    QLabel *emailLabel;
    QLineEdit *emailField;

    QLabel *passwordLabel;
    QLineEdit *passwordField;

    QPushButton *loginButton;
    QPushButton *signupButton;

    QGridLayout *formLayout;
};

#endif // FORMWIDGET_H
```

7.10 main.cpp File Reference

```
#include "mainwindow.h"
#include <QApplication>
#include <QString>
```

```
#include <QGraphicsDropShadowEffect>
#include <QPushButton>
```

Include dependency graph for main.cpp:



Functions

- int main (int argc, char ∗argv[ ])

7.10.1 Function Documentation

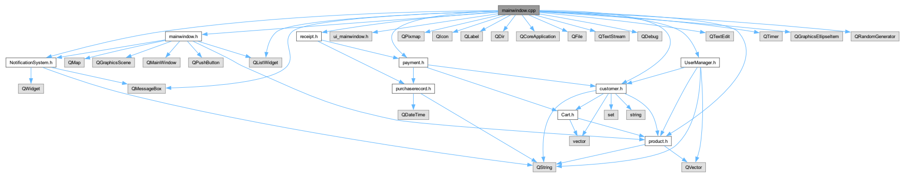
7.10.1.1 main()

```
int main (
            int argc,
            char ∗ argv[ ])
```

7.11 mainwindow.cpp File Reference

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "customer.h"
#include "product.h"
#include <QPixmap>
#include <QIcon>
#include <QListWidget>
#include <QLabel>
#include <QDir>
#include <QCoreApplication>
#include <QFile>
#include <QTextStream>
#include <QDebug>
#include <QMessageBox>
#include "payment.h"
#include "receipt.h"
#include <QTextEdit>
#include "NotificationSystem.h"
#include "UserManager.h"
#include "QTimer"
#include "QGraphicsEllipseItem"
```

```
#include "QRandomGenerator"
```
Include dependency graph for mainwindow.cpp:



Functions

- QVector< Product > getProductsVector ()

    Reads product data from a text file and returns a vector of products.
- QString sizeToString (Product::SIZE size)

    Converts a product size enum value to its corresponding string representation.

Variables

- UserManager userManager

7.11.1 Function Documentation

7.11.1.1 getProductsVector()

```
QVector< Product > getProductsVector ()
```

Reads product data from a text file and returns a vector of products.

This method parses the product information stored in a text file and creates `Product` objects for each entry. Each product's details, such as ID, image path, explanation, cost, and number of likes, are extracted and stored in a `QVector` of `Product` objects.

Returns

A `QVector` containing all the products read from the file.

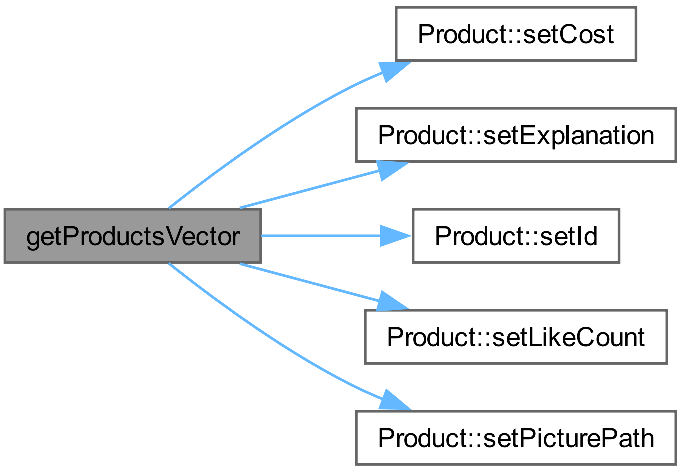
Note

The file path is hardcoded as `:/res/resources/product1.txt`, which is expected to be a resource file. If the file cannot be opened, an empty vector is returned, and a debug message is printed.

See also

Product

Here is the call graph for this function:



7.11.1.2 sizeToString()

```
QString sizeToString (
            Product::SIZE size)
```

Converts a product size enum value to its corresponding string representation.

This method maps the `Product::SIZE` enum values to their respective string equivalents, such as "XSMALL", "SMALL", "MEDIUM", "LARGE", and "XLARGE".

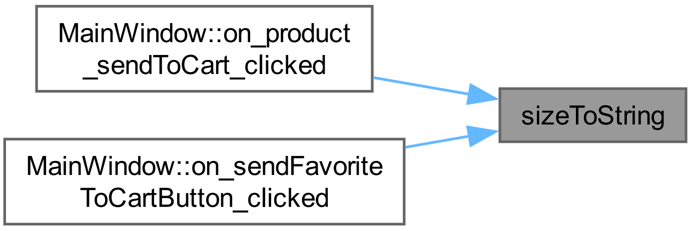
Parameters

size	The `Product::SIZE` enum value to be converted.

Returns

A `QString` representing the size as a human-readable string. Returns an empty string if the size does not match any known enum value.

Here is the caller graph for this function:


MainWindow::on_product _sendToCart_clicked
MainWindow::on_sendFavorite ToCartButton_clicked
sizeToString

7.11.2 Variable Documentation

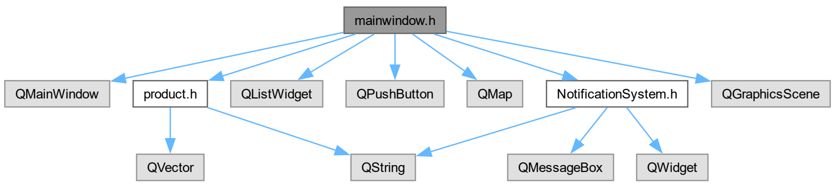
7.11.2.1 userManager

```
UserManager userManager
```

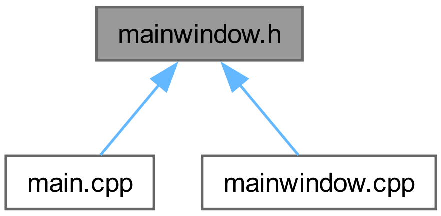
7.12 mainwindow.h File Reference

```
#include <QMainWindow>
#include "product.h"
#include <QListWidget>
#include <QPushButton>
#include <QMap>
#include "NotificationSystem.h"
#include <QGraphicsScene>
```

Include dependency graph for mainwindow.h:



This graph shows which files directly or indirectly include this file:



Classes

- class MainWindow

  Main application window for Cartify.

Namespaces

- namespace Ui

7.13 mainwindow.h

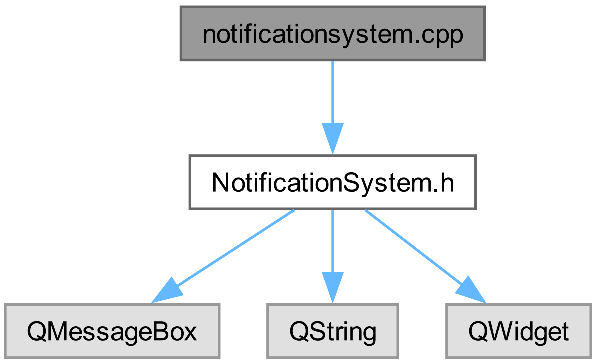
Go to the documentation of this file.

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "product.h"
#include <QListWidget>
#include <QPushButton>
#include <QMap>
#include "NotificationSystem.h"
#include <QGraphicsScene>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);

    ~MainWindow();

protected:
    void closeEvent(QCloseEvent *event) override;

private slots:
    void spinWheel();

    void applyDiscount(int percentage);

    void on_searchButton_clicked();

    void on_applyDiscountButtonClicked();

    void on_productButtonClicked(Product& product);

    void on_productButton_clicked();

    void on_likeButtonClicked();

    void on_submitCommentButtonClicked();

    void displayCommentsForProduct(const Product &product);

    void on_gotoSignUp_clicked();

    void on_signUpButton_clicked();

    void on_loginButton_clicked();

    void on_electronicsButton_clicked();

    void on_clothesButton_clicked();

    void on_mainScreenButton_clicked();

    void on_logoutButton_clicked();

    void on_product_favoriteButton_clicked();

    void on_product_sendToCart_clicked();

    void on_profileButton_clicked();

    void on_discardButton_clicked();

    void on_sendFavoriteToCartButton_clicked();

    void on_cartButton_clicked();

    void on_buyButton_clicked();

    void on_removeButton_clicked();

    void on_sizeComboBox_currentTextChanged(const QString &arg1);

    void on_pass_to_loginPage_clicked();

    void setProductButton(QPushButton *button, const QString &picturePath);

    void setupProductButtons();

    void showDiscountColors();

    void on_giftWrapCheckBox_toggled(bool checked);

private:
    Ui::MainWindow *ui;
    QListWidget *commentListWidget;
    NotificationSystem *notificationSystem;
    QVector<Product> products;
    QVector<QPushButton *> productButtons;
    QMap<QPushButton *, Product *> buttonProductMap;
    Product* currentProduct;

    QGraphicsScene *scene;
    QGraphicsEllipseItem *wheel;
    QTimer *timer;
    int currentAngle;
    int targetAngle;
    QStringList wheelRewards;
    double currentSpeed;
    double minSpeed;
    double deceleration;
    QList<QGraphicsPathItem*> slices;
    QMap<int, QString> colorMap;
    QMap<QString, int> discountMap;
    QTimer *blinkTimer;
    int blinkCount;
    bool blinkState;
    int chosenSliceIndex;
    int currentDiscount;
    bool discountApplied;

    const double giftWrapFee = 10.0;

    void connectProductButtons();
};

#endif // MAINWINDOW_H
```

7.14 notificationsystem.cpp File Reference

```
#include "NotificationSystem.h"
```

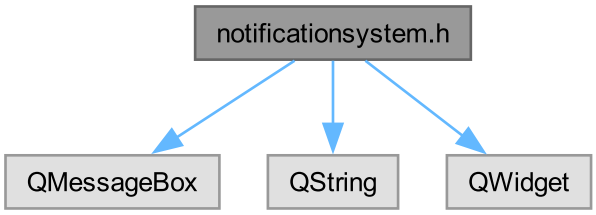
Include dependency graph for notificationsystem.cpp:



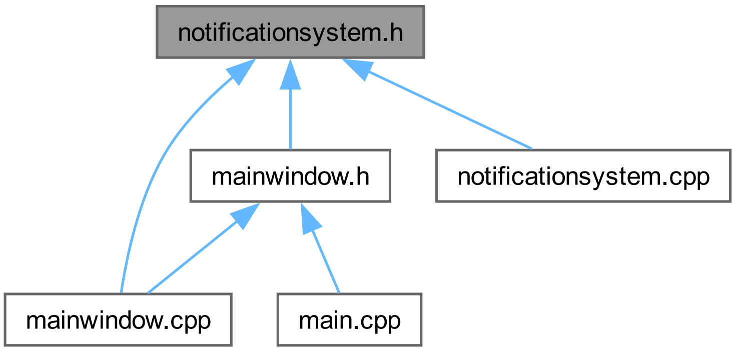
7.15 notificationsystem.h File Reference

```
#include <QMessageBox>
#include <QString>
#include <QWidget>
```

Include dependency graph for notificationsystem.h:



This graph shows which files directly or indirectly include this file:



Classes

- class NotificationSystem

    The NotificationSystem class provides an interface for displaying informational, warning, and error messages in a GUI application.

7.16 notificationsystem.h

Go to the documentation of this file.

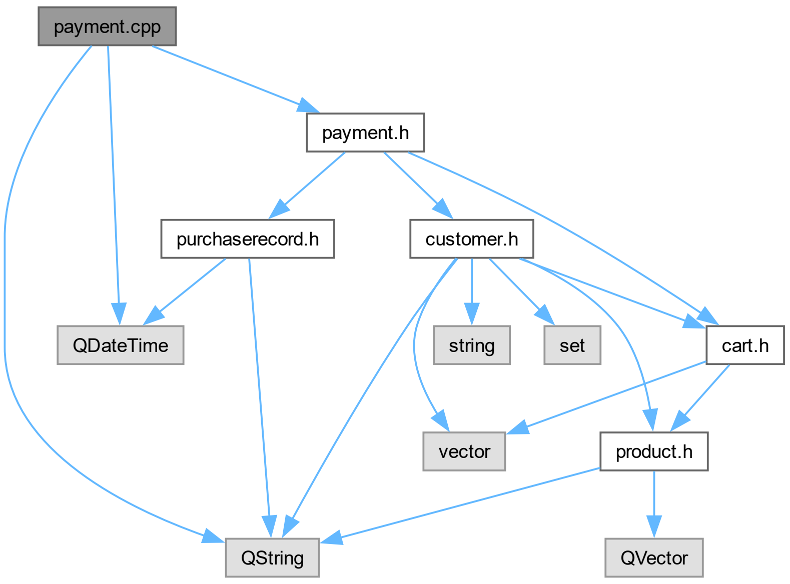
```
#ifndef NOTIFICATIONSYSTEM_H
#define NOTIFICATIONSYSTEM_H

#include <QMessageBox>
#include <QString>
#include <QWidget>

class NotificationSystem {
public:
    explicit NotificationSystem(QWidget *parent = nullptr);

    void showInfo(const QString &title, const QString &message);

    void showWarning(const QString &title, const QString &message);

    void showError(const QString &title, const QString &message);

private:
    QWidget *parentWidget;
};

#endif // NOTIFICATIONSYSTEM_H
```

7.17 payment.cpp File Reference

```
#include "payment.h"
#include <QDateTime>
```

```
#include <QString>
```

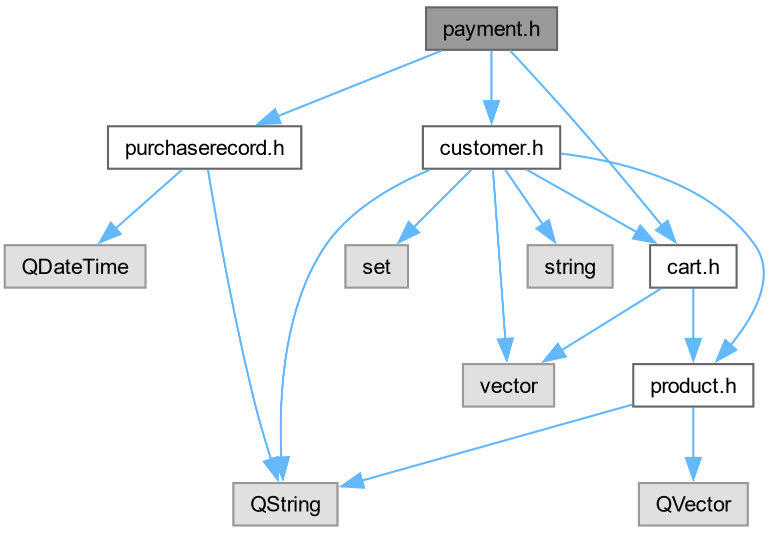
Include dependency graph for payment.cpp:



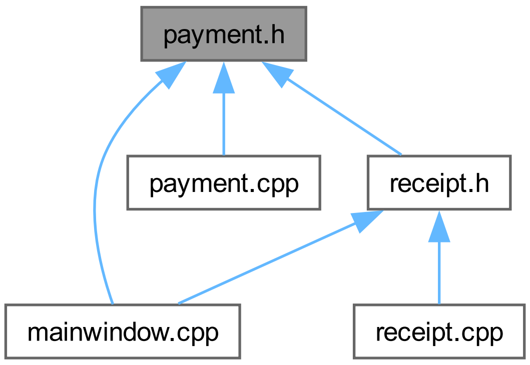
7.18 payment.h File Reference

```
#include "purchaserecord.h"
#include "cart.h"
#include "customer.h"
```

Include dependency graph for payment.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Payment

  Represents a payment process, including discounts and customer information.

Enumerations

- enum Discount { NoDiscount , D10 , D20 , D50 }

  Enum representing available discount types.

7.18.1 Enumeration Type Documentation

7.18.1.1 Discount

```
enum Discount
```

Enum representing available discount types.

Enumerator

NoDiscount	No discount applied.
D10	10% discount.
D20	20% discount.
D50	50% discount.

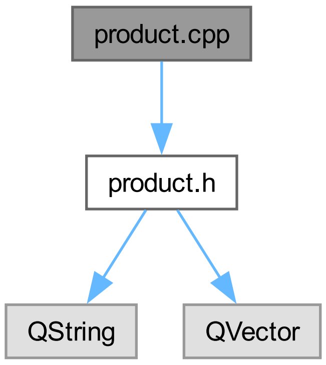
7.19 payment.h

Go to the documentation of this file.

```
#ifndef PAYMENT_H
#define PAYMENT_H

#include "purchaserecord.h"
#include "cart.h"
#include "customer.h"

enum Discount {
    NoDiscount,
    D10,
    D20,
    D50
};

class Payment : public PurchaseRecord {
private:
    Customer customer;
    Cart cart;
    Discount discount;

public:
    Payment(Customer customer, Cart cart, Discount discount);

    void setDiscount(Discount discount);

    Discount getDiscount() const;

    void setCustomer(const Customer& customer);

    Customer getCustomer() const;

    void setCart(const Cart& cart);

    Cart getCart() const;

    double grandTotal() const;

    double total() const;

    double applyDiscount() const;

    std::string discountPercentage() const;

    void applyDiscountCode(const QString &code);

    QString getRecordDetails() const override;
};

#endif // PAYMENT_H
```

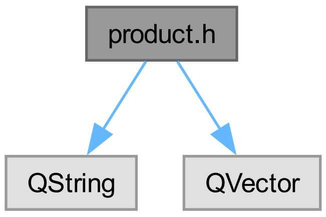
7.20 product.cpp File Reference

```
#include "product.h"
```
Include dependency graph for product.cpp:

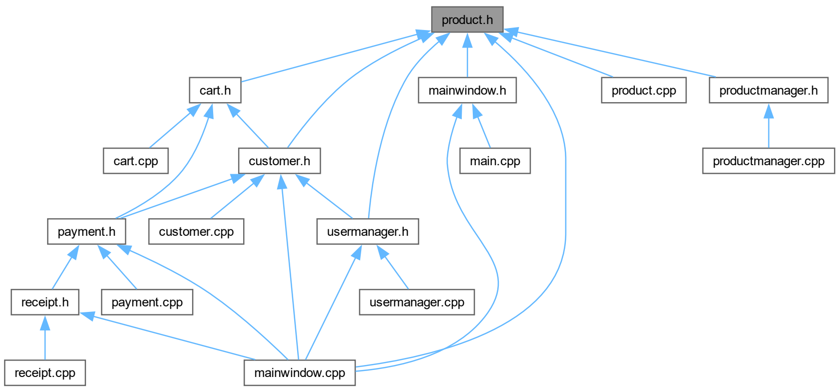


7.21 product.h File Reference

```
#include <QString>
#include <QVector>
```
Include dependency graph for product.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Product

  Represents a product in the Cartify system.

7.22 product.h

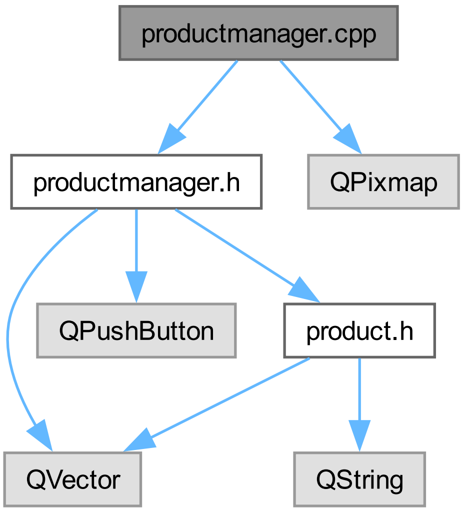
Go to the documentation of this file.

```
#ifndef PRODUCT_H
#define PRODUCT_H

#include <QString>
#include <QVector>

class Product {

private:
    int id;
    QString picturePath;
    QString explanation;
    double cost;
    int likeCount;
    QVector<QString> comments;

public:
    enum class SIZE {
        XSMALL,
        SMALL,
        MEDIUM,
        LARGE,
        XLARGE
    };

    SIZE selectedSize;

    Product();

    Product(int id, QString picturePath, QString explanation, double cost, int likeCount);

    Product(const Product& temp);

    int getId() const;

    void setId(int value);

    QString getPicturePath() const;

    void setPicturePath(const QString& value);

    QString getExplanation() const;

    void setExplanation(const QString& value);

    double getCost() const;

    void setCost(double value);

    int getLikeCount() const;

    void setLikeCount(int value);

    QVector<QString> getComments() const;

    void setComments(const QVector<QString>& value);

    void addComment(const QString &comment);

    void likeProduct();

    void unlikeProduct();

    SIZE getSelectedSize() const;

    void setSelectedSize(SIZE size);

    QString getSizeString() const;

    QString toQString() const;
};

#endif // PRODUCT_H
```

7.23 productmanager.cpp File Reference

```
#include "productmanager.h"
#include <QPixmap>
```

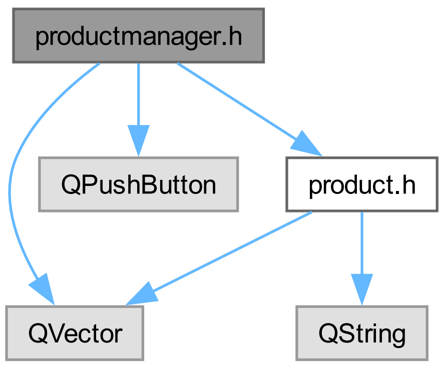
Include dependency graph for productmanager.cpp:



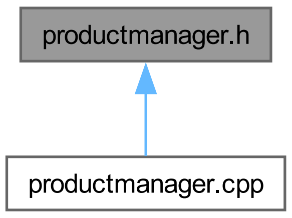
7.24 productmanager.h File Reference

```
#include <QVector>
#include <QPushButton>
#include "product.h"
```

Include dependency graph for productmanager.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ProductManager

  Manages products and their interactions within the Cartify system.

7.25 productmanager.h

Go to the documentation of this file.

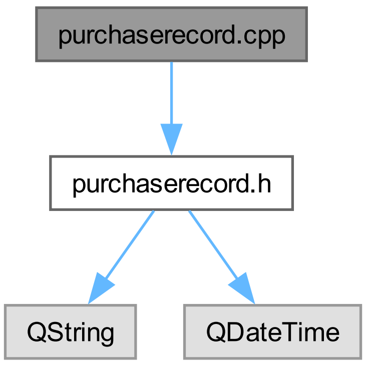
```
00001 #ifndef PRODUCTMANAGER_H
```

```
#define PRODUCTMANAGER_H

#include <QVector>
#include <QPushButton>
#include "product.h"



class ProductManager {
public:
    ProductManager(const QVector<Product> &products);

    void setupProductButtons(const QVector<QPushButton *> &buttons, int startIndex = 0);

    Product* getProductByIndex(int index);

private:
    QVector<Product> products;
};

#endif // PRODUCTMANAGER_H
```

7.26 purchaserecord.cpp File Reference

`#include "purchaserecord.h"`

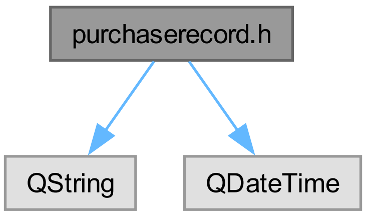
Include dependency graph for purchaserecord.cpp:



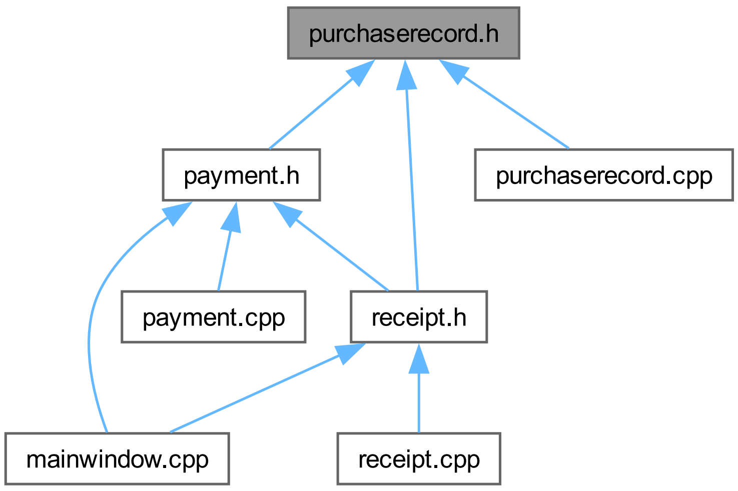
7.27 purchaserecord.h File Reference

```
#include <QString>
#include <QDateTime>
```

Include dependency graph for purchaserecord.h:



This graph shows which files directly or indirectly include this file:



Classes

- class PurchaseRecord

  Represents a generic purchase record in the Cartify system.

7.28 purchaserecord.h

Go to the documentation of this file.

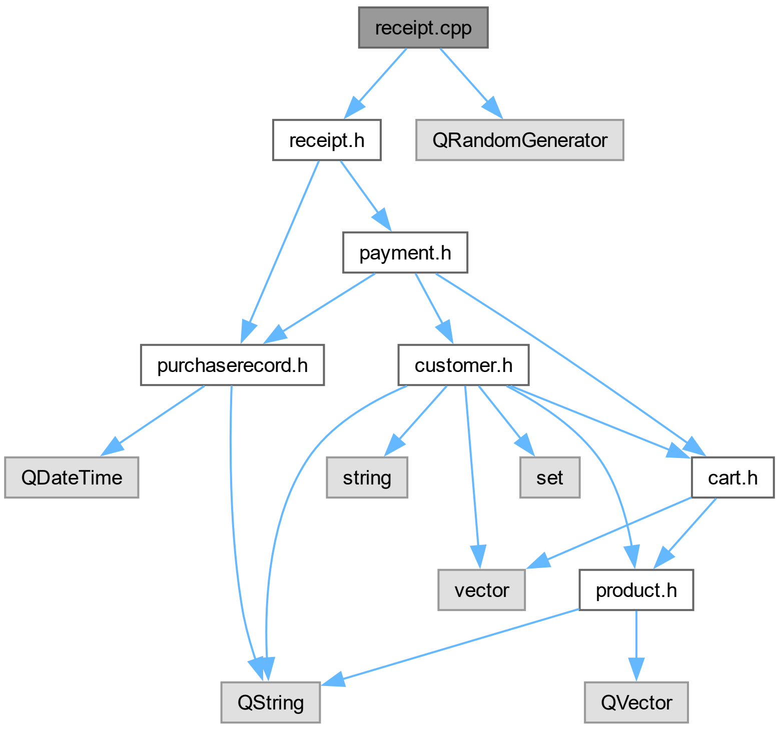
```
#ifndef PURCHASERECORD_H
#define PURCHASERECORD_H

#include <QString>
```

```
#include <QDateTime>

class PurchaseRecord {
protected:
    QDateTime purchaseDate;
    double totalAmount;

public:
    PurchaseRecord();

    PurchaseRecord(QDateTime date, double amount);

    virtual QString getRecordDetails() const = 0;

    QDateTime getPurchaseDate() const;

    void setPurchaseDate(const QDateTime& date);

    double getTotalAmount() const;

    void setTotalAmount(double amount);

    virtual ~PurchaseRecord() = default;
};

#endif // PURCHASERECORD_H
```

7.29 receipt.cpp File Reference

```
#include "receipt.h"
#include <QRandomGenerator>
```

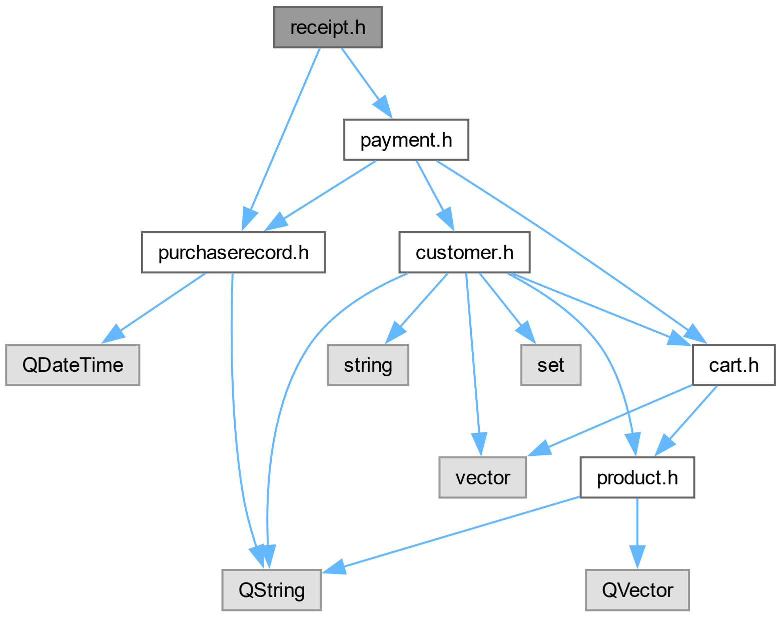
Include dependency graph for receipt.cpp:



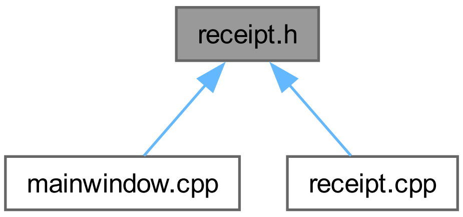
7.30 receipt.h File Reference

```
#include "purchaserecord.h"
#include "payment.h"
```

Include dependency graph for receipt.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Receipt

  Represents a receipt for a completed purchase in the Cartify system.

7.31 receipt.h

Go to the documentation of this file.

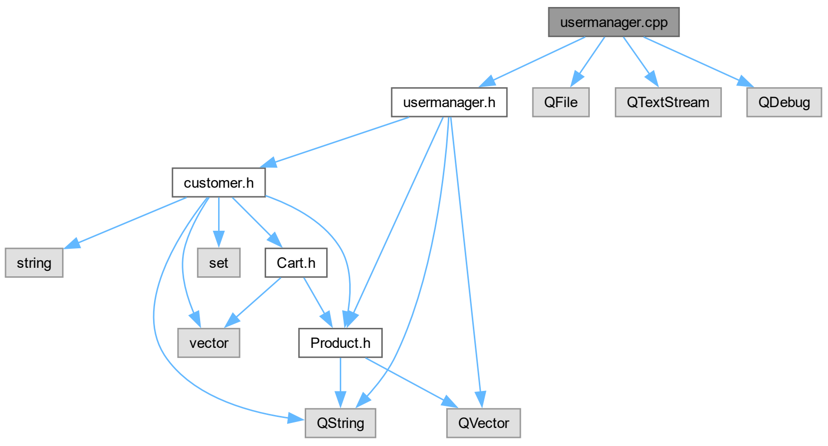
```
#ifndef RECEIPT_H
#define RECEIPT_H

#include "purchaserecord.h"
#include "payment.h"

class Receipt : public PurchaseRecord {
    Payment payment;

public:
    Receipt(const Payment& payment);

    int addPoint();

    QString orderNo() const;

    QString toString() const;

    QString getRecordDetails() const override;
};

#endif // RECEIPT_H
```

7.32 usermanager.cpp File Reference

Implementation of the UserManager class.

```
#include "usermanager.h"
#include <QFile>
#include <QTextStream>
#include <QDebug>
```

Include dependency graph for usermanager.cpp:



7.32.1 Detailed Description

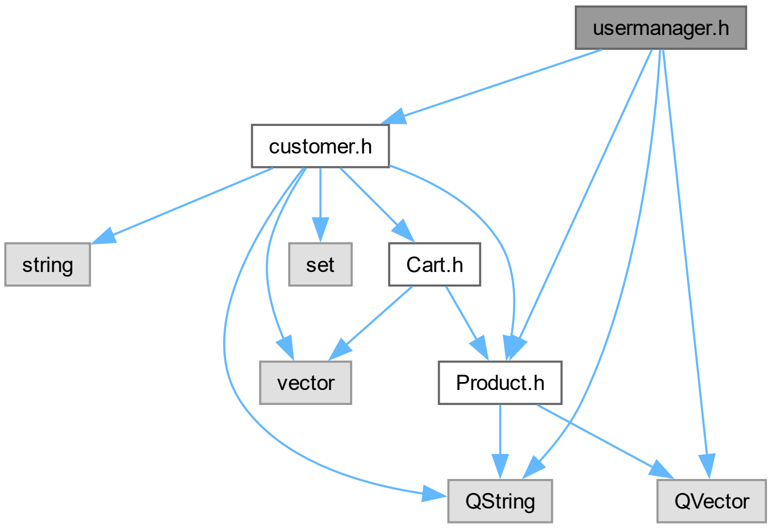
Implementation of the UserManager class.

This file provides the implementation of user authentication, registration, and management functionalities.

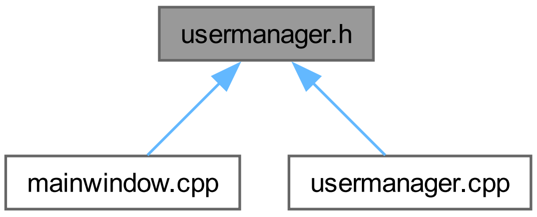
7.33 usermanager.h File Reference

```
#include "customer.h"
#include "product.h"
#include <QString>
#include <QVector>
```

Include dependency graph for usermanager.h:



This graph shows which files directly or indirectly include this file:



Classes

- class UserManager

  Manages user accounts and authentication in the Cartify system.

7.34 usermanager.h

Go to the documentation of this file.

```
#ifndef USERMANAGER_H
#define USERMANAGER_H

#include "customer.h"
#include "product.h"
#include <QString>
#include <QVector>

class UserManager {
private:
    Customer currentUser;

public:
    UserManager();

    Customer& getCurrentUser();

    bool login(const QString &email, const QString &password, const QVector<Product> &products);

    bool isRegistered(const QString &email);

    void signUp(const QString &name, const QString &surname, const QString &email, const QString
&password);
};

#endif // USERMANAGER_H
```

Index