

Cartify

Generated by Doxygen 1.12.0



<b>1 Namespace Index</b>	<b>1</b>
1.1 Namespace List	1
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Namespace Documentation</b>	<b>9</b>
5.1 Ui Namespace Reference	9
<b>6 Class Documentation</b>	<b>11</b>
6.1 Cart Class Reference	11
6.1.1 Detailed Description	11
6.1.2 Constructor & Destructor Documentation	12
6.1.2.1 Cart()	12
6.1.3 Member Function Documentation	12
6.1.3.1 addProduct()	12
6.1.3.2 clearCart()	12
6.1.3.3 getProducts()	13
6.1.3.4 removeProductById()	13
6.1.4 Member Data Documentation	14
6.1.4.1 products	14
6.2 Customer Class Reference	14
6.2.1 Detailed Description	16
6.2.2 Constructor & Destructor Documentation	16
6.2.2.1 Customer()	16
6.2.2.2 ~Customer()	17
6.2.3 Member Function Documentation	17
6.2.3.1 addCart()	17
6.2.3.2 addFavorite()	17
6.2.3.3 addPoint()	18
6.2.3.4 addPurchasedProduct()	19
6.2.3.5 getCart()	19
6.2.3.6 getCartObject()	20
6.2.3.7 getEmail()	21
6.2.3.8 getFavorites()	21
6.2.3.9 getName()	22
6.2.3.10 getPoints()	22
6.2.3.11 getProductType()	22

6.2.3.12 getSurname()	23
6.2.3.13 hasLikedProduct()	23
6.2.3.14 hasPurchasedProduct()	23
6.2.3.15 isCartEmpty()	24
6.2.3.16 likeProduct()	25
6.2.3.17 myPoints()	25
6.2.3.18 removeCart()	26
6.2.3.19 removeFavorite()	26
6.2.3.20 setFavorites()	26
6.2.3.21 setName()	27
6.2.3.22 setPassword()	27
6.2.3.23 setPoints()	28
6.2.3.24 setProductType()	28
6.2.3.25 setSurname()	28
6.2.4 Member Data Documentation	28
6.2.4.1 cart	28
6.2.4.2 email	29
6.2.4.3 favorites	29
6.2.4.4 likedProductIds	29
6.2.4.5 name	29
6.2.4.6 password	29
6.2.4.7 point	29
6.2.4.8 previousOrders	29
6.2.4.9 productType	29
6.2.4.10 purchasedProducts	30
6.2.4.11 surname	30
6.3 FormWidget Class Reference	30
6.3.1 Detailed Description	31
6.3.2 Constructor & Destructor Documentation	31
6.3.2.1 FormWidget()	31
6.3.3 Member Data Documentation	32
6.3.3.1 emailField	32
6.3.3.2 emailLabel	32
6.3.3.3 formLayout	32
6.3.3.4 loginButton	32
6.3.3.5 passwordField	32
6.3.3.6 passwordLabel	32
6.3.3.7 signupButton	33
6.4 MainWindow Class Reference	33
6.4.1 Detailed Description	36
6.4.2 Constructor & Destructor Documentation	36
6.4.2.1 MainWindow()	36

6.4.2.2 ~MainWindow()	37
6.4.3 Member Function Documentation	37
6.4.3.1 applyDiscount	37
6.4.3.2 closeEvent()	37
6.4.3.3 connectProductButtons()	38
6.4.3.4 displayCommentsForProduct	38
6.4.3.5 on_applyDiscountButtonClicked	39
6.4.3.6 on_buyButton_clicked	40
6.4.3.7 on_cartButton_clicked	41
6.4.3.8 on_clothesButton_clicked	41
6.4.3.9 on_discardButton_clicked	42
6.4.3.10 on_electronicsButton_clicked	42
6.4.3.11 on_giftWrapCheckBox_toggled	42
6.4.3.12 on_gotoSignUp_clicked	43
6.4.3.13 on_likeButtonClicked	43
6.4.3.14 on_loginButton_clicked	44
6.4.3.15 on_logoutButton_clicked	45
6.4.3.16 on_mainScreenButton_clicked	46
6.4.3.17 on_pass_to_loginPage_clicked	46
6.4.3.18 on_product_favoriteButton_clicked	46
6.4.3.19 on_product_sendToCart_clicked	47
6.4.3.20 on_productButton_clicked	47
6.4.3.21 on_productButtonClicked	48
6.4.3.22 on_profileButton_clicked	49
6.4.3.23 on_removeButton_clicked	50
6.4.3.24 on_searchButton_clicked	51
6.4.3.25 on_sendFavoriteToCartButton_clicked	52
6.4.3.26 on_signUpButton_clicked	52
6.4.3.27 on_sizeComboBox_currentTextChanged	53
6.4.3.28 on_submitCommentButtonClicked	54
6.4.3.29 setProductButton	55
6.4.3.30 setupProductButtons	55
6.4.3.31 showDiscountColors	56
6.4.3.32 spinWheel	56
6.4.4 Member Data Documentation	57
6.4.4.1 blinkCount	57
6.4.4.2 blinkState	57
6.4.4.3 blinkTimer	57
6.4.4.4 buttonProductMap	57
6.4.4.5 chosenSliceIndex	57
6.4.4.6 colorMap	57
6.4.4.7 commentListWidget	57

6.4.4.8 currentAngle	57
6.4.4.9 currentDiscount	57
6.4.4.10 currentProduct	58
6.4.4.11 currentSpeed	58
6.4.4.12 deceleration	58
6.4.4.13 discountApplied	58
6.4.4.14 discountMap	58
6.4.4.15 giftWrapFee	58
6.4.4.16 minSpeed	58
6.4.4.17 notificationSystem	58
6.4.4.18 productButtons	58
6.4.4.19 products	59
6.4.4.20 scene	59
6.4.4.21 slices	59
6.4.4.22 targetAngle	59
6.4.4.23 timer	59
6.4.4.24 ui	59
6.4.4.25 wheel	59
6.4.4.26 wheelRewards	59
6.5 NotificationSystem Class Reference	60
6.5.1 Detailed Description	60
6.5.2 Constructor & Destructor Documentation	60
6.5.2.1 NotificationSystem()	60
6.5.3 Member Function Documentation	60
6.5.3.1 showError()	60
6.5.3.2 showInfo()	61
6.5.3.3 showWarning()	62
6.5.4 Member Data Documentation	62
6.5.4.1 parentWidget	62
6.6 Payment Class Reference	63
6.6.1 Detailed Description	65
6.6.2 Constructor & Destructor Documentation	65
6.6.2.1 Payment()	65
6.6.3 Member Function Documentation	65
6.6.3.1 applyDiscount()	65
6.6.3.2 applyDiscountCode()	66
6.6.3.3 discountPercentage()	66
6.6.3.4 getCart()	67
6.6.3.5 getCustomer()	67
6.6.3.6 getDiscount()	67
6.6.3.7 getRecordDetails()	68
6.6.3.8 grandTotal()	68

6.6.3.9 setCart()	69
6.6.3.10 setCustomer()	69
6.6.3.11 setDiscount()	69
6.6.3.12 total()	70
6.6.4 Member Data Documentation	70
6.6.4.1 cart	70
6.6.4.2 customer	70
6.6.4.3 discount	71
6.7 Product Class Reference	71
6.7.1 Detailed Description	72
6.7.2 Member Enumeration Documentation	73
6.7.2.1 SIZE	73
6.7.3 Constructor & Destructor Documentation	74
6.7.3.1 Product() [1/3]	74
6.7.3.2 Product() [2/3]	74
6.7.3.3 Product() [3/3]	74
6.7.4 Member Function Documentation	75
6.7.4.1 addComment()	75
6.7.4.2 getComments()	75
6.7.4.3 getCost()	76
6.7.4.4 getExplanation()	76
6.7.4.5 getId()	77
6.7.4.6 getLikeCount()	77
6.7.4.7 getPicturePath()	78
6.7.4.8 getSelectedSize()	78
6.7.4.9 getSizeString()	79
6.7.4.10 likeProduct()	79
6.7.4.11 setComments()	79
6.7.4.12 setCost()	80
6.7.4.13 setExplanation()	80
6.7.4.14 setId()	81
6.7.4.15 setLikeCount()	82
6.7.4.16 setPicturePath()	82
6.7.4.17 setSelectedSize()	83
6.7.4.18 toQString()	83
6.7.4.19 unlikeProduct()	84
6.7.5 Member Data Documentation	84
6.7.5.1 comments	84
6.7.5.2 cost	84
6.7.5.3 explanation	84
6.7.5.4 id	84
6.7.5.5 likeCount	84

6.7.5.6 picturePath . . . . .	84
6.7.5.7 selectedSize . . . . .	85
6.8 ProductManager Class Reference . . . . .	85
6.8.1 Detailed Description . . . . .	85
6.8.2 Constructor & Destructor Documentation . . . . .	85
6.8.2.1 ProductManager() . . . . .	85
6.8.3 Member Function Documentation . . . . .	86
6.8.3.1 getProductByIndex() . . . . .	86
6.8.3.2 setupProductButtons() . . . . .	86
6.8.4 Member Data Documentation . . . . .	87
6.8.4.1 products . . . . .	87
6.9 PurchaseRecord Class Reference . . . . .	88
6.9.1 Detailed Description . . . . .	89
6.9.2 Constructor & Destructor Documentation . . . . .	89
6.9.2.1 PurchaseRecord() [1/2] . . . . .	89
6.9.2.2 PurchaseRecord() [2/2] . . . . .	89
6.9.2.3 ~PurchaseRecord() . . . . .	89
6.9.3 Member Function Documentation . . . . .	89
6.9.3.1 getPurchaseDate() . . . . .	89
6.9.3.2 getRecordDetails() . . . . .	90
6.9.3.3 getTotalAmount() . . . . .	90
6.9.3.4 setPurchaseDate() . . . . .	90
6.9.3.5 setTotalAmount() . . . . .	90
6.9.4 Member Data Documentation . . . . .	91
6.9.4.1 purchaseDate . . . . .	91
6.9.4.2 totalAmount . . . . .	91
6.10 Receipt Class Reference . . . . .	91
6.10.1 Detailed Description . . . . .	93
6.10.2 Constructor & Destructor Documentation . . . . .	93
6.10.2.1 Receipt() . . . . .	93
6.10.3 Member Function Documentation . . . . .	94
6.10.3.1 addPoint() . . . . .	94
6.10.3.2 getRecordDetails() . . . . .	95
6.10.3.3 orderNo() . . . . .	95
6.10.3.4 toString() . . . . .	96
6.10.4 Member Data Documentation . . . . .	96
6.10.4.1 payment . . . . .	96
6.11 UserManager Class Reference . . . . .	97
6.11.1 Detailed Description . . . . .	97
6.11.2 Constructor & Destructor Documentation . . . . .	98
6.11.2.1 UserManager() . . . . .	98
6.11.3 Member Function Documentation . . . . .	98



6.11.3.1 getCurrentUser()	98
6.11.3.2 isRegistered()	99
6.11.3.3 login()	100
6.11.3.4 signUp()	101
6.11.4 Member Data Documentation	102
6.11.4.1 currentUser	102
<b>7 File Documentation</b>	<b>103</b>
7.1 cart.cpp File Reference	103
7.2 cart.h File Reference	103
7.3 cart.h	105
7.4 customer.cpp File Reference	105
7.5 customer.h File Reference	106
7.5.1 Enumeration Type Documentation	107
7.5.1.1 ProductType	107
7.6 customer.h	107
7.7 FormWidget.cpp File Reference	109
7.8 FormWidget.h File Reference	109
7.9 FormWidget.h	110
7.10 main.cpp File Reference	110
7.10.1 Function Documentation	111
7.10.1.1 main()	111
7.11 mainwindow.cpp File Reference	111
7.11.1 Function Documentation	112
7.11.1.1 getProductsVector()	112
7.11.1.2 sizeToString()	113
7.11.2 Variable Documentation	114
7.11.2.1 userManager	114
7.12 mainwindow.h File Reference	114
7.13 mainwindow.h	115
7.14 notificationsystem.cpp File Reference	117
7.15 notificationsystem.h File Reference	117
7.16 notificationsystem.h	118
7.17 payment.cpp File Reference	118
7.18 payment.h File Reference	119
7.18.1 Enumeration Type Documentation	120
7.18.1.1 Discount	120
7.19 payment.h	121
7.20 product.cpp File Reference	122
7.21 product.h File Reference	122
7.22 product.h	123
7.23 productmanager.cpp File Reference	124

7.24 productmanager.h File Reference . . . . .	125
7.25 productmanager.h . . . . .	125
7.26 purchaserecord.cpp File Reference . . . . .	126
7.27 purchaserecord.h File Reference . . . . .	126
7.28 purchaserecord.h . . . . .	127
7.29 receipt.cpp File Reference . . . . .	128
7.30 receipt.h File Reference . . . . .	129
7.31 receipt.h . . . . .	130
7.32 usermanager.cpp File Reference . . . . .	130
7.32.1 Detailed Description . . . . .	130
7.33 usermanager.h File Reference . . . . .	131
7.34 usermanager.h . . . . .	132
<b>Index</b>	<b>133</b>

# Chapter 1

## Namespace Index

### 1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">Ui</a> . . . . .	9
------------------------------	---



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

Cart . . . . .	11
Customer . . . . .	14
NotificationSystem . . . . .	60
Product . . . . .	71
ProductManager . . . . .	85
PurchaseRecord . . . . .	88
Payment . . . . .	63
Receipt . . . . .	91
QMainWindow	
MainWindow . . . . .	33
QWidget	
FormWidget . . . . .	30
UserManager . . . . .	97



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">Cart</a>	Represents a shopping cart that holds a collection of products . . . . .	11
<a href="#">Customer</a>	Represents a customer in the Cartify system . . . . .	14
<a href="#">FormWidget</a>	A widget that provides a login and sign-up form interface . . . . .	30
<a href="#">MainWindow</a>	Main application window for Cartify . . . . .	33
<a href="#">NotificationSystem</a>	Interface for displaying informational, warning, and error messages in a GUI application . . . .	60
<a href="#">Payment</a>	Represents a payment process, including discounts and customer information . . . . .	63
<a href="#">Product</a>	Represents a product in the Cartify system . . . . .	71
<a href="#">ProductManager</a>	Manages products and their interactions within the Cartify system . . . . .	85
<a href="#">PurchaseRecord</a>	Represents a generic purchase record in the Cartify system . . . . .	88
<a href="#">Receipt</a>	Represents a receipt for a completed purchase in the Cartify system . . . . .	91
<a href="#">UserManager</a>	Manages user accounts and authentication in the Cartify system . . . . .	97





## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

<a href="#">cart.cpp</a>	103
<a href="#">cart.h</a>	103
<a href="#">customer.cpp</a>	105
<a href="#">customer.h</a>	106
<a href="#">FormWidget.cpp</a>	109
<a href="#">FormWidget.h</a>	109
<a href="#">main.cpp</a>	110
<a href="#">mainwindow.cpp</a>	111
<a href="#">mainwindow.h</a>	114
<a href="#">notificationssystem.cpp</a>	117
<a href="#">notificationssystem.h</a>	117
<a href="#">payment.cpp</a>	118
<a href="#">payment.h</a>	119
<a href="#">product.cpp</a>	122
<a href="#">product.h</a>	122
<a href="#">productmanager.cpp</a>	124
<a href="#">productmanager.h</a>	125
<a href="#">purchaserecord.cpp</a>	126
<a href="#">purchaserecord.h</a>	126
<a href="#">receipt.cpp</a>	128
<a href="#">receipt.h</a>	129
<a href="#">usermanager.cpp</a>	
Implementation of the <a href="#">UserManager</a> class	130
<a href="#">usermanager.h</a>	131



## 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()

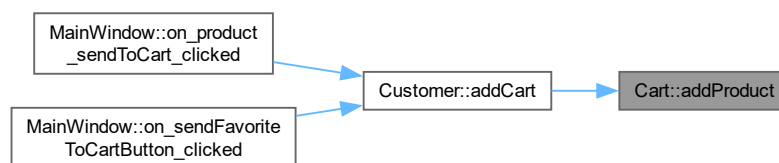
```
void Cart::addProduct (  
    const Product & product)
```

Adds a product to the cart.

#### Parameters

<i>product</i>	The product to be added to the cart.
----------------	--------------------------------------

Here is the caller graph for this function:



### 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()

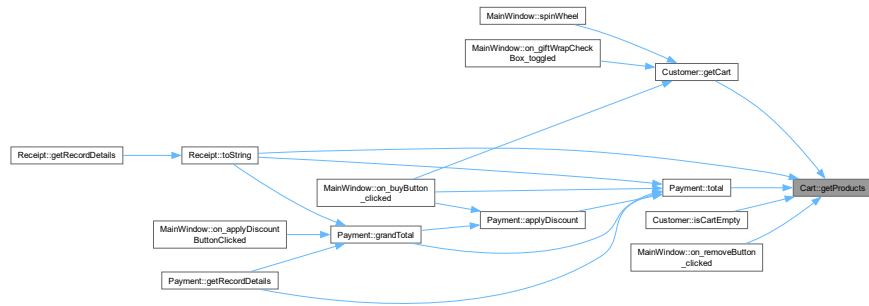
```
const std::vector< Product > & Cart::getProducts () const
```

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:



### 6.1.3.4 removeProductById()

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

Removes a product from the cart by its ID.

#### Parameters

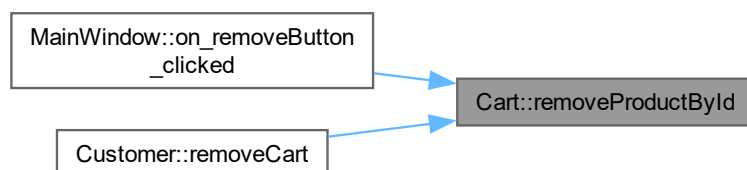
<i>productId</i>	The ID of the product to remove.
------------------	----------------------------------

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

#### Parameters

<i>productId</i>	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:

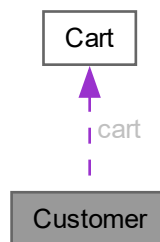
- [cart.h](#)
- [cart.cpp](#)

## 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

<i>name</i>	The customer's first name.
<i>surname</i>	The customer's surname.
<i>productType</i>	The customer's preferred product type.
<i>email</i>	The customer's email address.
<i>password</i>	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.

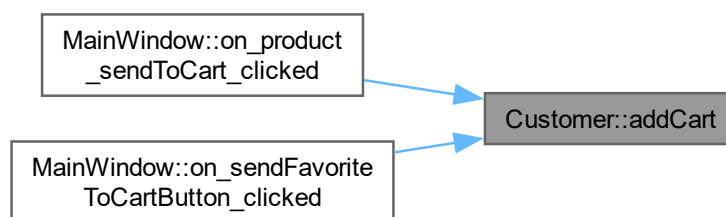
#### Parameters

<i>prod</i>	The product to add.
-------------	---------------------

Here is the call graph for this function:



Here is the caller graph for this function:



### 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**

<i>prod</i>	The product to add.
-------------	---------------------

**Returns**

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

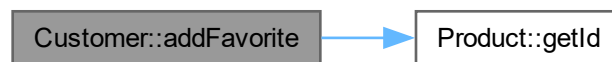
**Parameters**

<i>prod</i>	The product to add.
-------------	---------------------

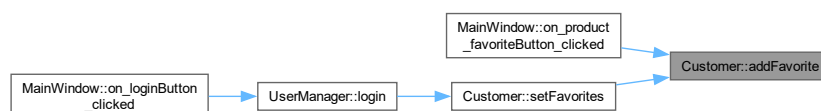
**Returns**

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:

**6.2.3.3 addPoint()**

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

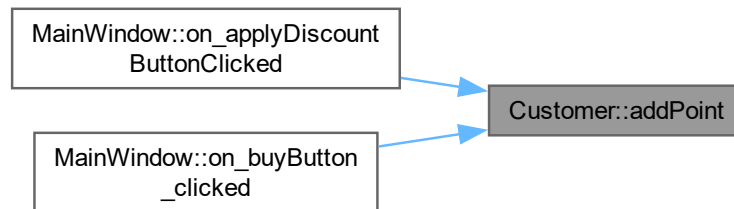
Adds points to the customer's account.

**Parameters**

<i>amount</i>	The amount of points to add.
---------------	------------------------------

---

Here is the caller graph for this function:



#### 6.2.3.4 addPurchasedProduct()

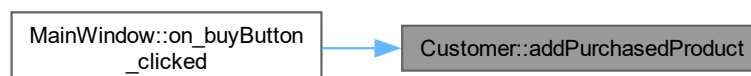
```
void Customer::addPurchasedProduct (  
    const Product & product)
```

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

##### Parameters

<i>product</i>	The product to add.
----------------	---------------------

Here is the caller graph for this function:



#### 6.2.3.5 getCart()

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

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

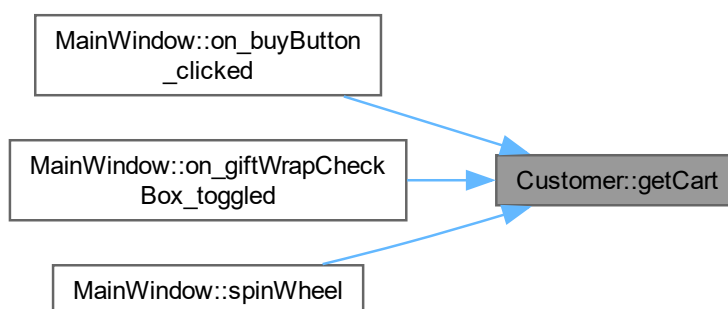
**Returns**

A vector of products currently in the cart.

Here is the call graph for this function:



Here is the caller graph for this function:

**6.2.3.6 getCartObject()**

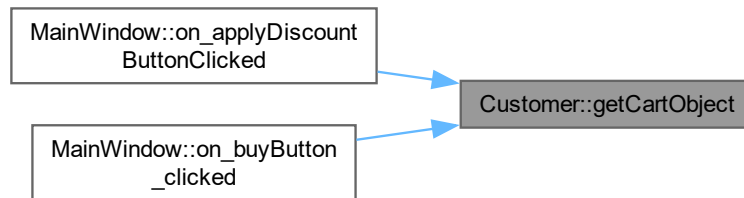
```
Cart Customer::getCartObject ()
```

Retrieves the customer's cart object.

**Returns**

A [Cart](#) object representing the customer's shopping cart.

Here is the caller graph for this function:

**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()**

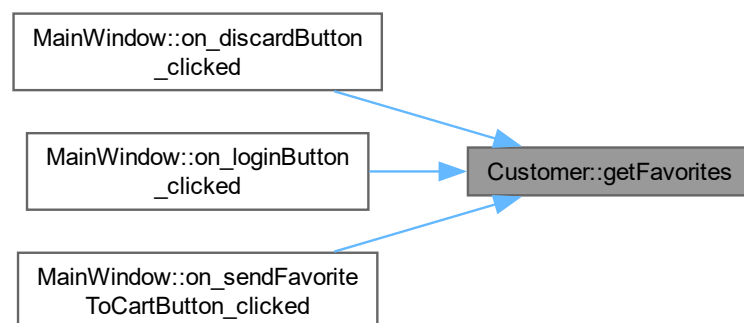
```
QVector< Product > Customer::getFavorites ()
```

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:



### 6.2.3.9 getName()

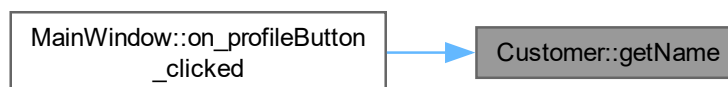
```
QString Customer::getName () const [inline]
```

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()**

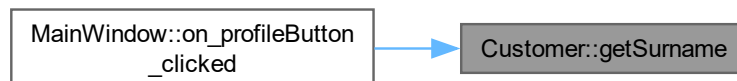
```
QString Customer::getSurname () const [inline]
```

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.

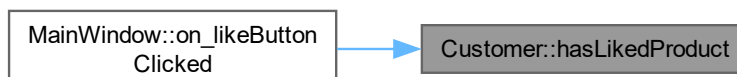
**Parameters**

<i>productId</i>	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**

<i>product</i> ↔ <i>Id</i>	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:

**6.2.3.15 isCartEmpty()**

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

Checks if the cart is empty.

**Parameters**

<i>prod</i>	A product to check against.
-------------	-----------------------------

**Returns**

True if the cart is empty, false otherwise.

**Parameters**

<i>prod</i>	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

<i>productId</i>	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()

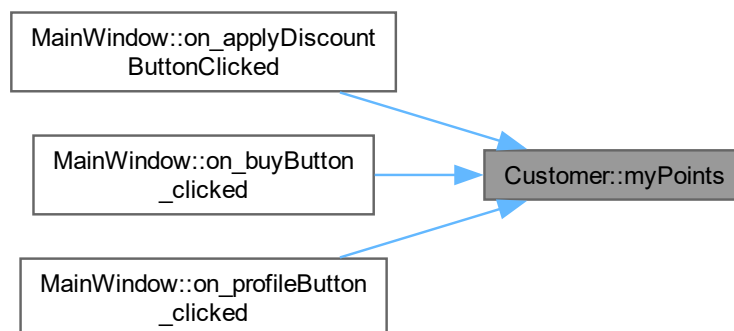
```
int Customer::myPoints () const
```

Retrieves the customer's current points.

#### Returns

The number of points the customer has.

Here is the caller graph for this function:



#### 6.2.3.18 removeCart()

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

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

##### Parameters

<i>id</i>	The ID of the product to remove.
-----------	----------------------------------

Here is the call graph for this function:



#### 6.2.3.19 removeFavorite()

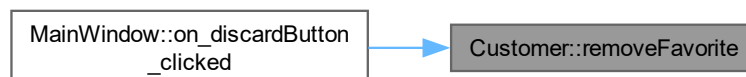
```
void Customer::removeFavorite (  
    int id)
```

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

##### Parameters

<i>id</i>	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

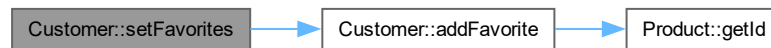
<i>favorites</i>	A comma-separated string of product IDs.
<i>products</i>	A QVector of available products.

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

## Parameters

<i>favorites</i>	A comma-separated string of product IDs.
<i>products</i>	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

<i>newName</i>	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**

<i>newPassword</i>	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**

<i>newPoints</i>	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**

<i>newProductType</i>	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**

<i>newSurname</i>	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:

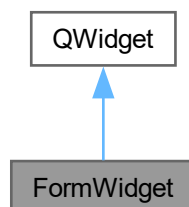
- [customer.h](#)
- [customer.cpp](#)

## 6.3 FormWidget Class Reference

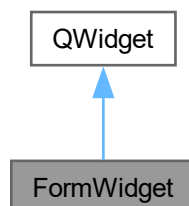
A widget that provides a login and sign-up form interface.

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

Inheritance diagram for FormWidget:



Collaboration diagram for FormWidget:





## Public Member Functions

- [QWidget](#) (QWidget \*parent=nullptr)  
*Constructs a [QWidget](#) 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 QWidget()

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

Constructs a [QWidget](#) object.

Constructs the [QWidget](#) and initializes the UI elements.

#### Parameters

<i>parent</i>	The parent widget. Defaults to nullptr.
---------------	-----------------------------------------

The [QWidget](#) 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**

<i>parent</i>	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:

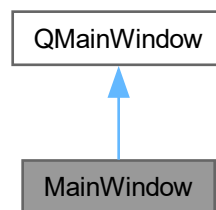
- [FormWidget.h](#)
- [FormWidget.cpp](#)

## 6.4 MainWindow Class Reference

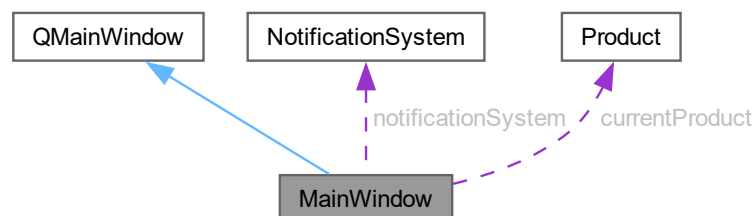
Main application window for Cartify.

```
#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

<i>parent</i>	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

<i>parent</i>	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

<i>percentage</i>	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

<i>percentage</i>	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

<i>event</i>	The <code>QCloseEvent</code> 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

<i>event</i>	Pointer to the <code>QCloseEvent</code> object that contains information about the close event.
--------------	-------------------------------------------------------------------------------------------------

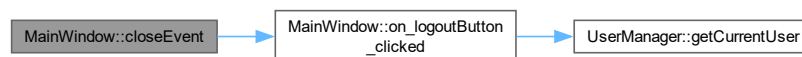
## Note

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:



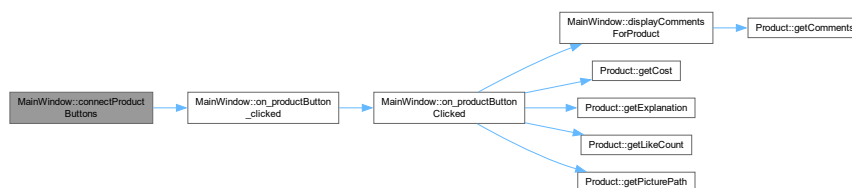
## 6.4.3.3 connectProductButtons()

```
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:



## 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

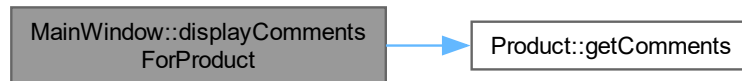
<i>product</i>	The product whose comments are displayed.
----------------	-------------------------------------------

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.

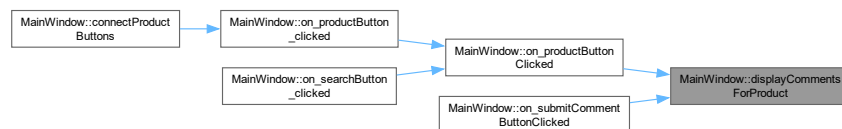
## Parameters

<i>product</i>	The <a href="#">Product</a> object whose comments are to be displayed.
----------------	------------------------------------------------------------------------

Here is the call graph for this function:



Here is the caller graph for this function:



#### 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.

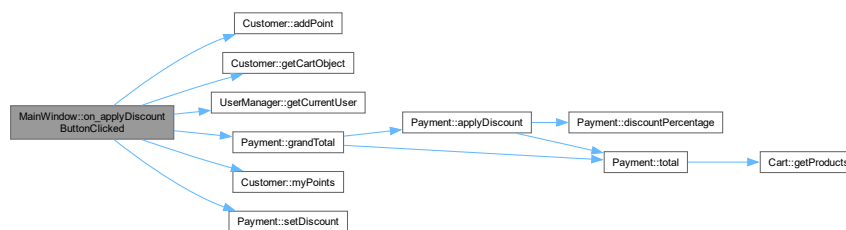
**Note**

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.

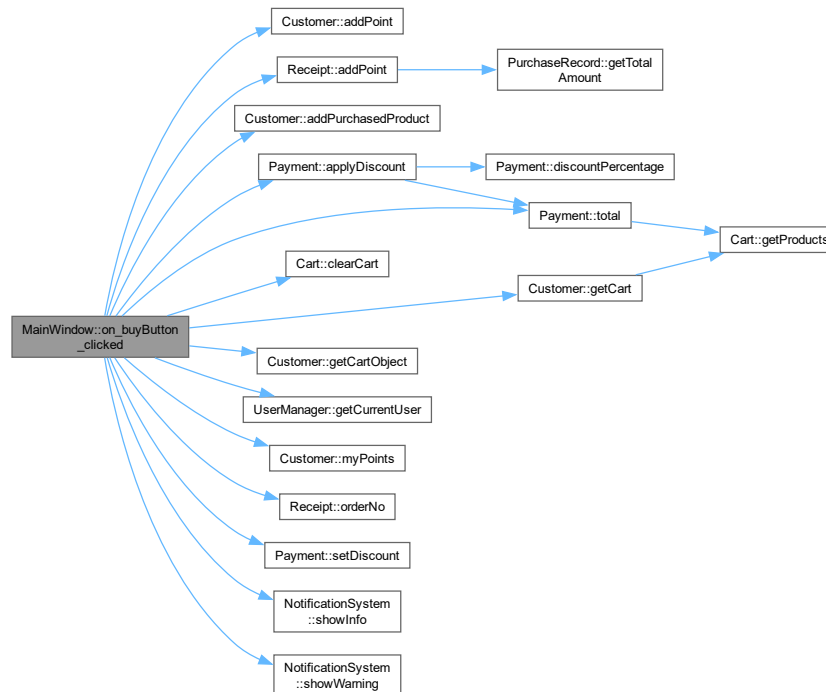
**Note**

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:



#### 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.

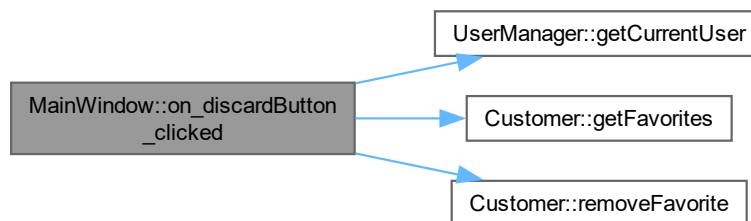
Handles the click event for the "Discard" button.

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:



#### 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

<i>checked</i>	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.

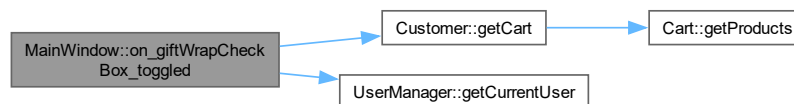
## Parameters

<i>checked</i>	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:



## 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**

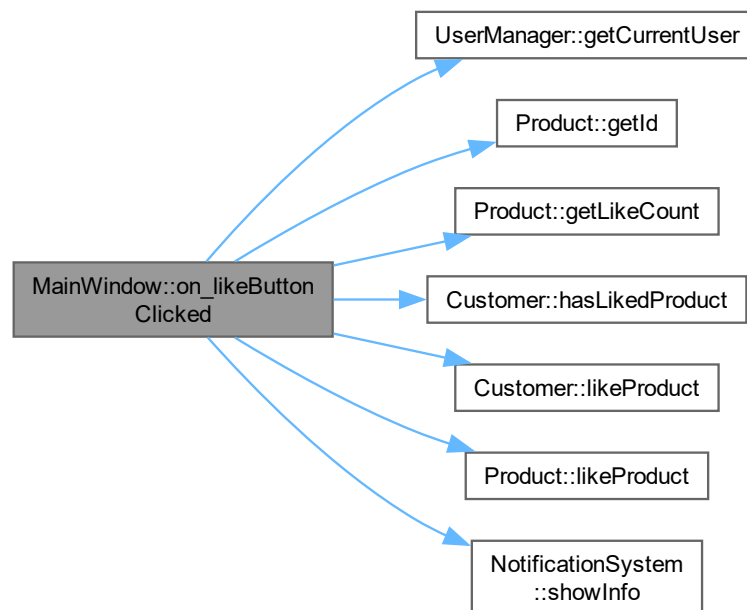
The method ensures a valid product is selected before proceeding with the action.

**See also**

[Customer::hasLikedProduct](#)

[Product::likeProduct](#)

Here is the call graph for this function:

**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:

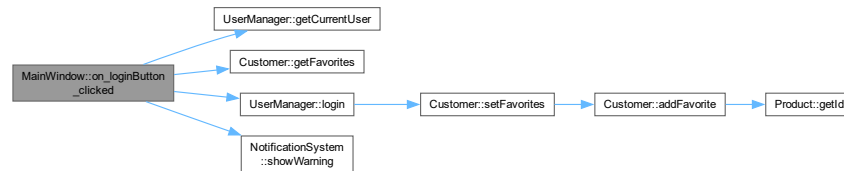
- The current user's favorites are retrieved and populated in the favorites combo box.
- The UI is navigated to the main application page.

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.

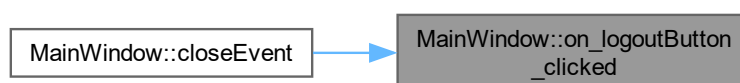
**Note**

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:



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.

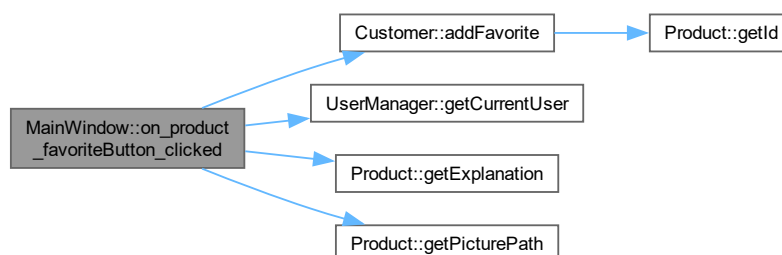
Handles the click event for the "Favorite Product" button.

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.

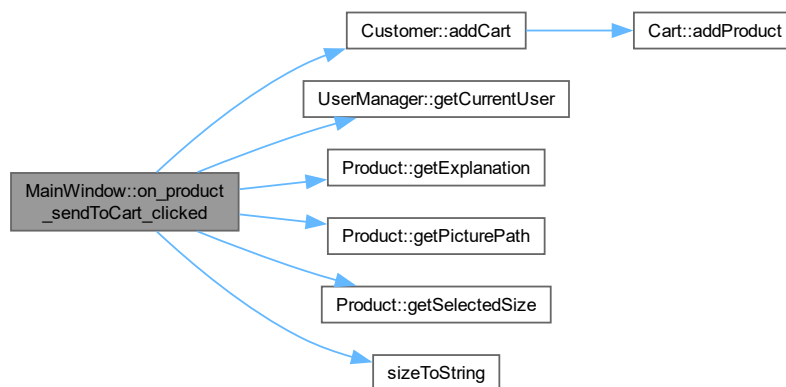
Handles the click event for the "Send to Cart" button.

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.

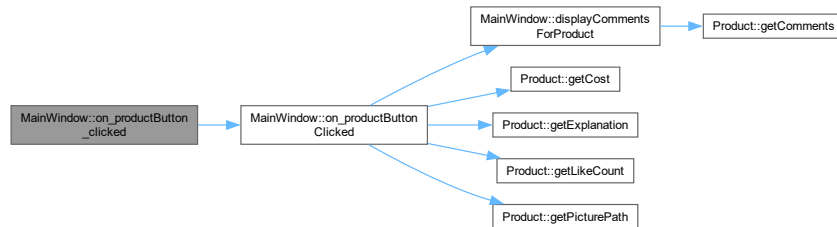
Handles the click event for a product button.

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.

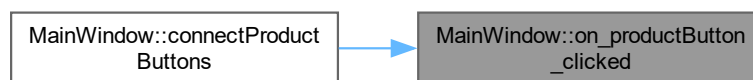
**Note**

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:



Here is the caller graph for this function:

**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**

<i>product</i>	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.

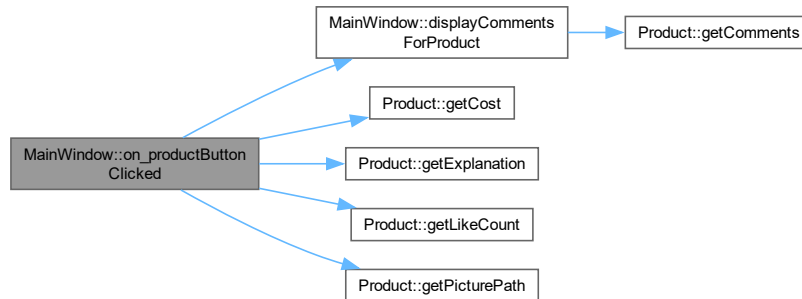
**Parameters**

<i>product</i>	Reference to the selected <code>Product</code> object.
----------------	--------------------------------------------------------

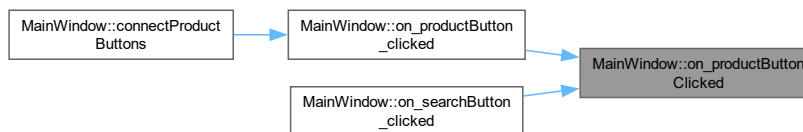
**Note**

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:

**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.

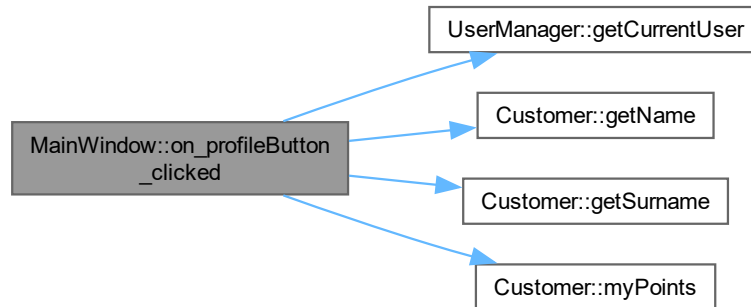
Updates the profile page with the user's information.

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 `userManager`.

Here is the call graph for this function:

**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.

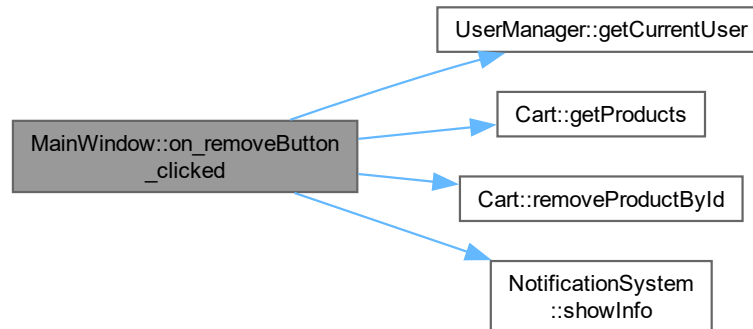
**Note**

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:



#### 6.4.3.24 on\_searchButton\_clicked

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

Handles the search button click event.

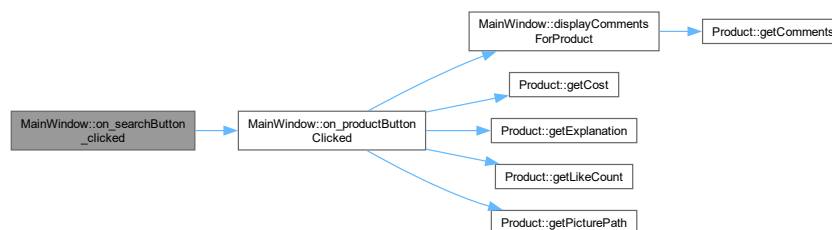
Searches for products based on the user's input and updates the GUI with the results.

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:



#### 6.4.3.25 on\_sendFavoriteToCartButton\_clicked

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

Sends a favorite product to the shopping cart.

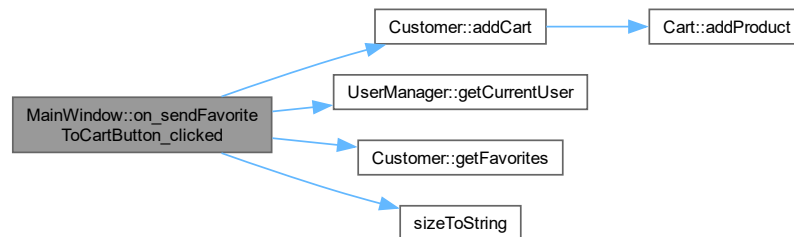
Handles the click event for the "Send Favorite to Cart" button.

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:



#### 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:

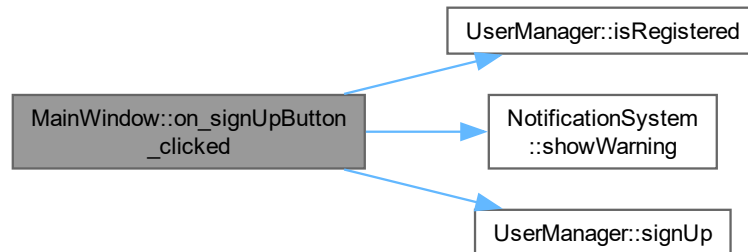
- Name and surname must be at least 3 characters long.
- Email must contain ".com" to be considered valid.
- Password must be at least 3 characters long.
- The email must not already be registered.

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:

**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**

<i>arg1</i>	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**

<i>arg1</i>	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:

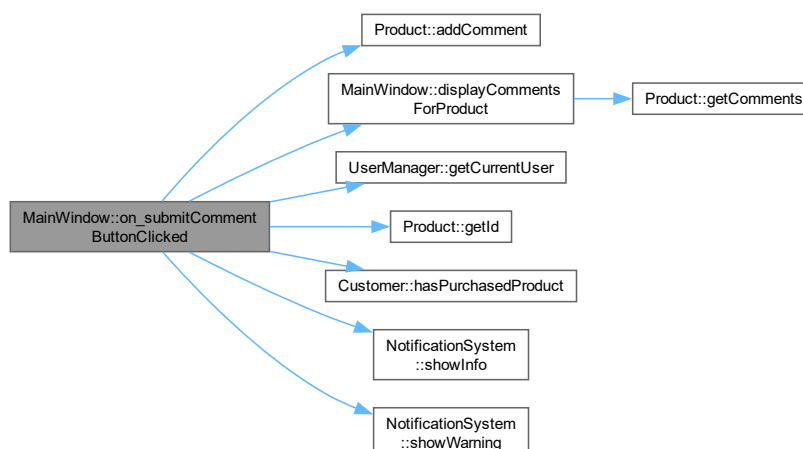
- The user must have purchased the product.
- The comment text must not be empty.

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:





#### 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

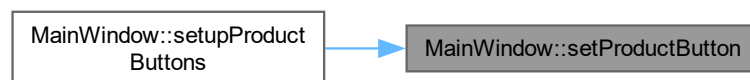
<i>button</i>	The QPushButton to configure.
<i>picturePath</i>	The path to the product's picture.

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

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

Here is the caller graph for this function:



#### 6.4.3.30 setupProductButtons

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

Initializes product buttons.

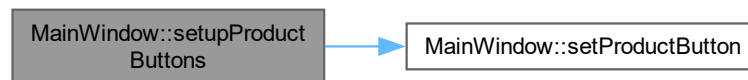
Initializes and configures product buttons for the UI.

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:

**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.

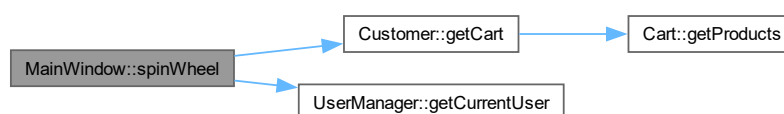
Initiates the spinning of the discount wheel.

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 <notificationssystem.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

<i>parent</i>	The parent widget for displaying message boxes. Defaults to nullptr.
---------------	----------------------------------------------------------------------

Initializes the [NotificationSystem](#) with a parent widget for displaying message boxes.

#### Parameters

<i>parent</i>	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

<i>title</i>	The title of the message box.
<i>message</i>	The error message to display.

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

## Parameters

<i>title</i>	The title of the message box.
<i>message</i>	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

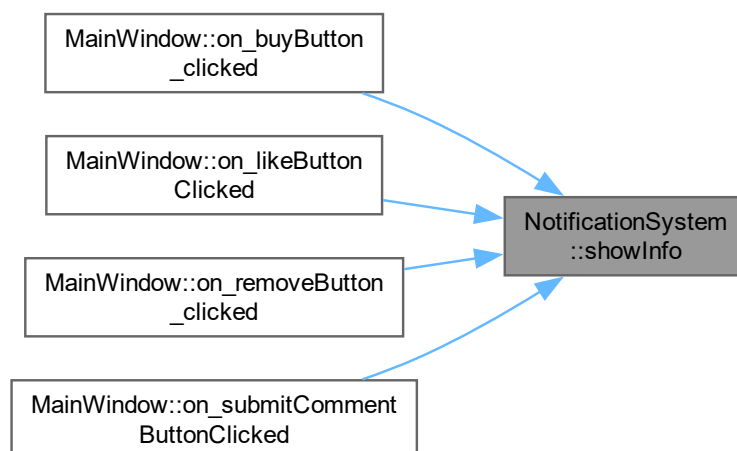
<i>title</i>	The title of the message box.
<i>message</i>	The informational message to display.

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

## Parameters

<i>title</i>	The title of the message box.
<i>message</i>	The informational message to display.

Here is the caller graph for this function:



### 6.5.3.3 showWarning()

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

Displays a warning message box.

#### Parameters

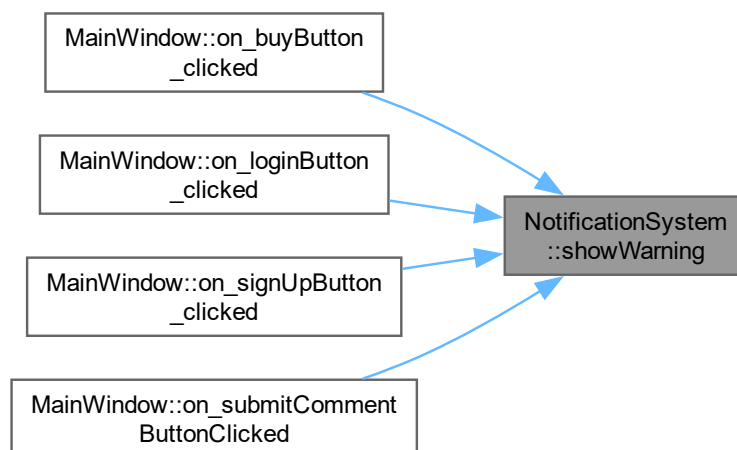
<i>title</i>	The title of the message box.
<i>message</i>	The warning message to display.

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

#### Parameters

<i>title</i>	The title of the message box.
<i>message</i>	The warning message to display.

Here is the caller graph for this function:



## 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](#)

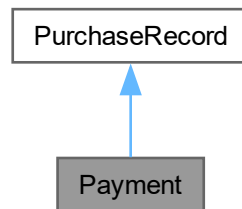


## 6.6 Payment Class Reference

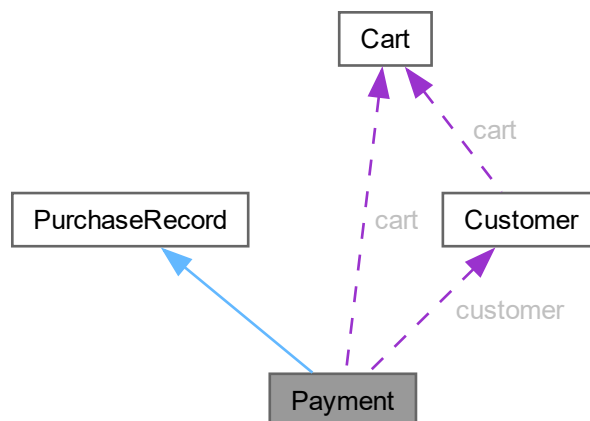
Represents a payment process, including discounts and customer information.

```
#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

<i>customer</i>	The customer making the payment.
<i>cart</i>	The shopping cart being purchased.
<i>discount</i>	The discount type applied to the payment.

### 6.6.3 Member Function Documentation

#### 6.6.3.1 applyDiscount()

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

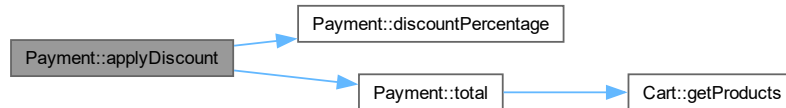
Calculates the discount amount.

Calculates the discount amount based on the applied discount type.

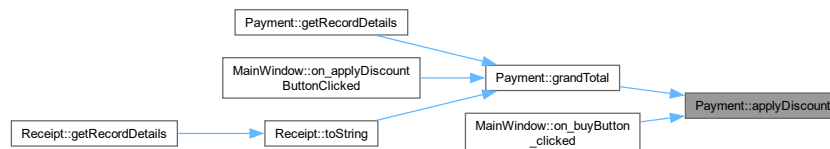
**Returns**

The total discount applied.

Here is the call graph for this function:



Here is the caller graph for this function:

**6.6.3.2 applyDiscountCode()**

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

Applies a discount code to the payment.

**Parameters**

<i>code</i>	The discount code as a QString.
-------------	---------------------------------

**6.6.3.3 discountPercentage()**

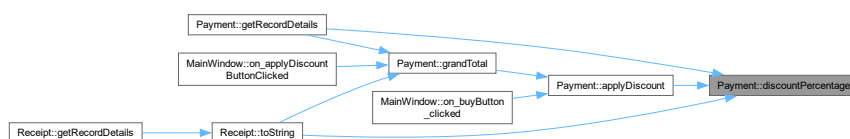
```
std::string Payment::discountPercentage () const
```

Converts the discount type to a string representation.

**Returns**

A string describing the discount percentage.

Here is the caller graph for this function:



#### 6.6.3.4 getCart()

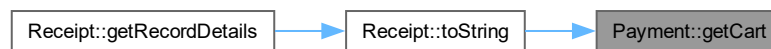
```
Cart Payment::getCart () const
```

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:



### 6.6.3.8 grandTotal()

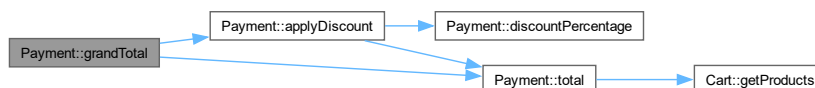
```
double Payment::grandTotal () const
```

Calculates the grand total after applying discounts.

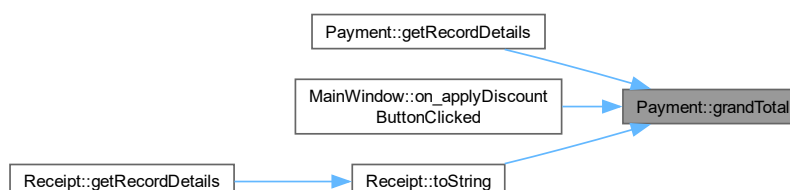
#### Returns

The total price after discounts.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.6.3.9 setCart()

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

Sets the cart associated with the payment.

#### Parameters

<i>cart</i>	The cart object to set.
-------------	-------------------------

### 6.6.3.10 setCustomer()

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

Sets the customer associated with the payment.

#### Parameters

<i>customer</i>	The customer object to set.
-----------------	-----------------------------

### 6.6.3.11 setDiscount()

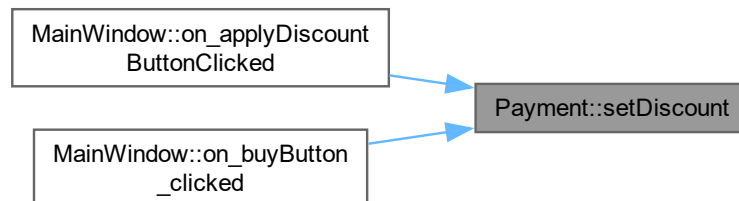
```
void Payment::setDiscount (
    Discount discount)
```

Sets the discount type.

#### Parameters

<i>discount</i>	The discount to apply.
-----------------	------------------------

Here is the caller graph for this function:



### 6.6.3.12 total()

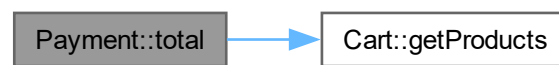
```
double Payment::total () const
```

Calculates the total cost of all items in the cart.

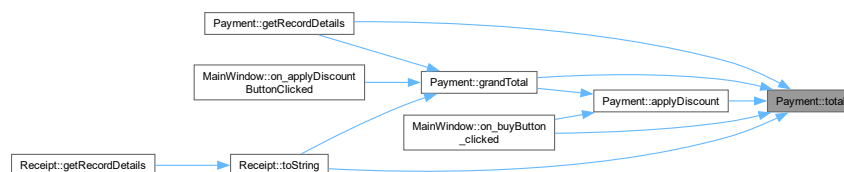
#### Returns

The total price without discounts.

Here is the call graph for this function:



Here is the caller graph for this function:



## 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**

<i>id</i>	The unique identifier for the product.
<i>picturePath</i>	The file path to the product's image.
<i>explanation</i>	The description of the product.
<i>cost</i>	The cost of the product.
<i>likeCount</i>	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**

<i>temp</i>	The product to copy.
-------------	----------------------

Copies the details of an existing product.

## Parameters

<i>temp</i>	The product to copy.
-------------	----------------------

## 6.7.4 Member Function Documentation

### 6.7.4.1 addComment()

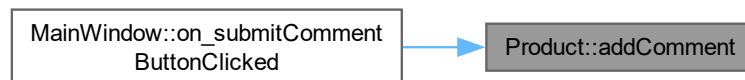
```
void Product::addComment (
    const QString & comment)
```

Adds a comment to the product.

## Parameters

<i>comment</i>	The comment to add.
----------------	---------------------

Here is the caller graph for this function:



### 6.7.4.2 getComments()

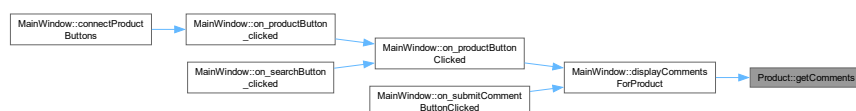
```
QVector< QString > Product::getComments () const
```

Retrieves the comments associated with the product.

## Returns

A QVector of comments.

Here is the caller graph for this function:



### 6.7.4.3 `getCost()`

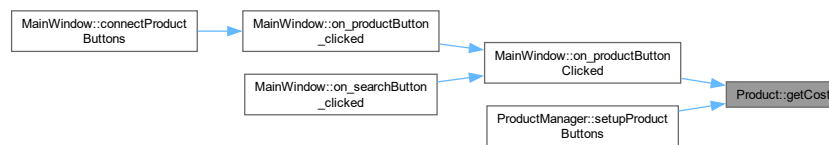
```
double Product::getCost () const
```

Retrieves the product's cost.

#### Returns

The cost of the product.

Here is the caller graph for this function:



### 6.7.4.4 `getExplanation()`

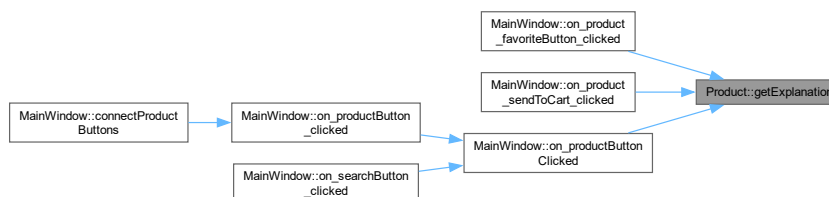
```
QString Product::getExplanation () const
```

Retrieves the product's description.

#### Returns

The description of the product.

Here is the caller graph for this function:



#### 6.7.4.5 getId()

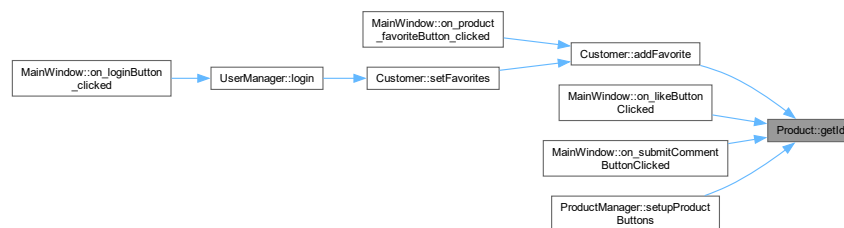
```
int Product::getId () const
```

Retrieves the product's ID.

##### Returns

The ID of the product.

Here is the caller graph for this function:



#### 6.7.4.6 getLikeCount()

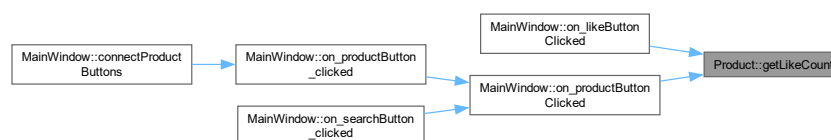
```
int Product::getLikeCount () const
```

Retrieves the number of likes for the product.

##### Returns

The number of likes.

Here is the caller graph for this function:



#### 6.7.4.7 getPicturePath()

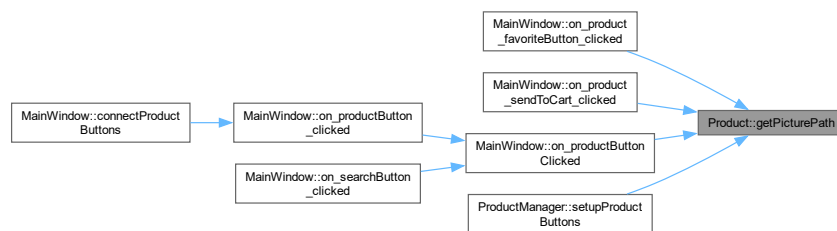
```
QString Product::getPicturePath () const
```

Retrieves the picture path of the product.

##### Returns

The file path to the product's image.

Here is the caller graph for this function:



#### 6.7.4.8 getSelectedSize()

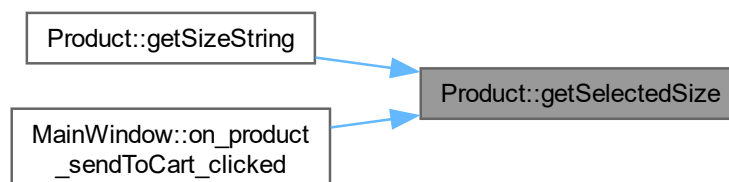
```
Product::SIZE Product::getSelectedSize () const
```

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:

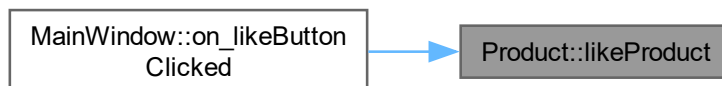


#### 6.7.4.10 likeProduct()

```
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

<i>value</i>	A QVector of comments to set.
--------------	-------------------------------

**6.7.4.12 setCost()**

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

Sets the product's cost.

## Parameters

<i>value</i>	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

<i>value</i>	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**

<i>value</i>	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**

<i>value</i>	The like count to set.
--------------	------------------------

Here is the caller graph for this function:

**6.7.4.16 setPicturePath()**

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

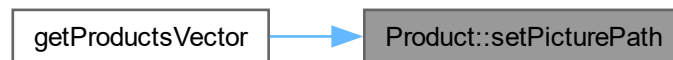
Sets the picture path for the product.

**Parameters**

<i>value</i>	The file path to set.
--------------	-----------------------

---

Here is the caller graph for this function:



#### 6.7.4.17 setSelectedSize()

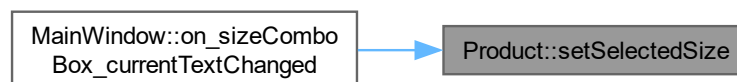
```
void Product::setSelectedSize (  
    SIZE size)
```

Sets the selected size for the product.

##### Parameters

<i>size</i>	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**

<i>products</i>	A QVector containing the products to manage.
-----------------	----------------------------------------------

Initializes the [ProductManager](#) with the provided QVector of [Product](#) objects.

**Parameters**

<i>products</i>	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**

<i>index</i>	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**

<i>index</i>	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

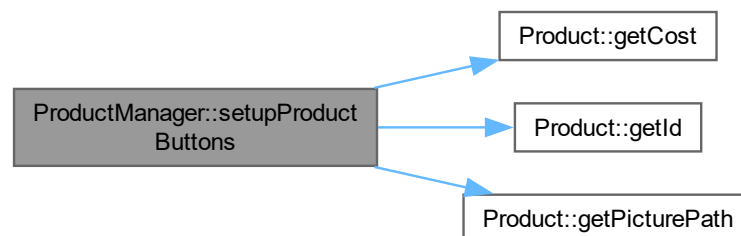
<i>buttons</i>	A QVector of QPushButton pointers to configure.
<i>startIndex</i>	The starting index in the product list. Defaults to 0.

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

<i>buttons</i>	A QVector of QPushButton pointers to configure.
<i>startIndex</i>	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:

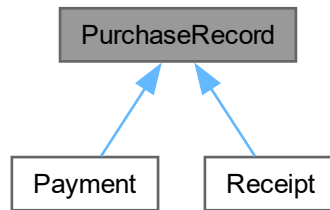
- [productmanager.h](#)
- [productmanager.cpp](#)

## 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

<i>date</i>	The date and time of the purchase.
<i>amount</i>	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:



### 6.9.3.4 setPurchaseDate()

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

Sets the date and time of the purchase.

#### Parameters

<i>date</i>	The date and time to set.
-------------	---------------------------

### 6.9.3.5 setTotalAmount()

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

Sets the total amount of the purchase.

## Parameters

<i>amount</i>	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:

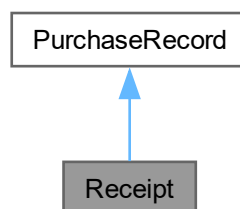
- [purchaserecord.h](#)
- [purchaserecord.cpp](#)

## 6.10 Receipt Class Reference

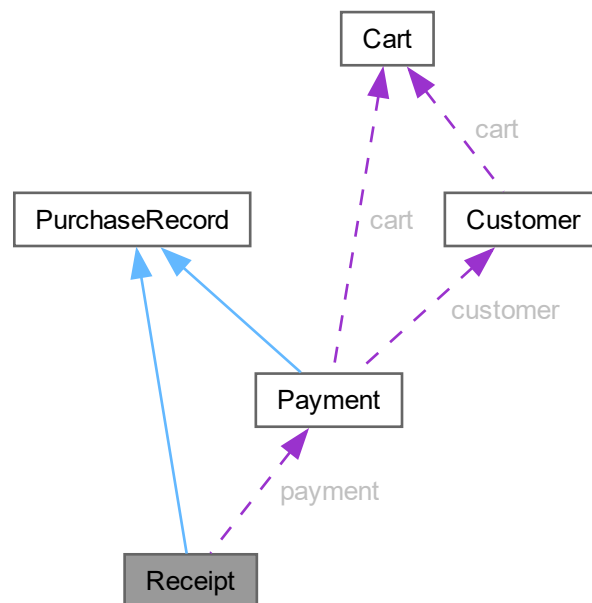
Represents a receipt for a completed purchase in the Cartify system.

```
#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

<i>payment</i>	The <a href="#">Payment</a> object containing purchase details.
----------------	-----------------------------------------------------------------

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

#### Parameters

<i>payment</i>	The <a href="#">Payment</a> 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.

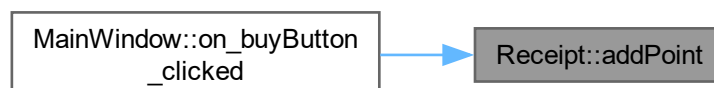
##### Returns

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.

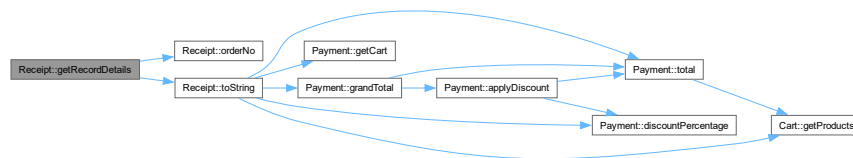
Combines the order number and receipt information into a single string.

#### 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
```

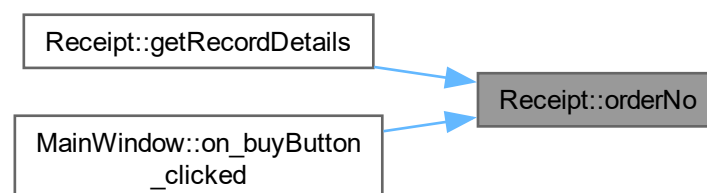
Generates a unique order number for the receipt.

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
```

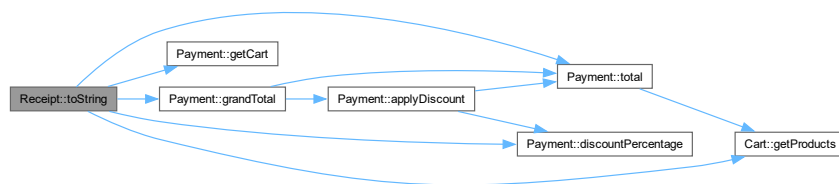
Converts the receipt details to a formatted string.

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:

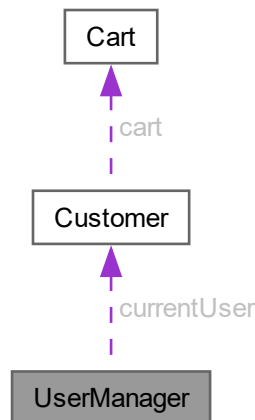
- [receipt.h](#)
- [receipt.cpp](#)

## 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 ()`

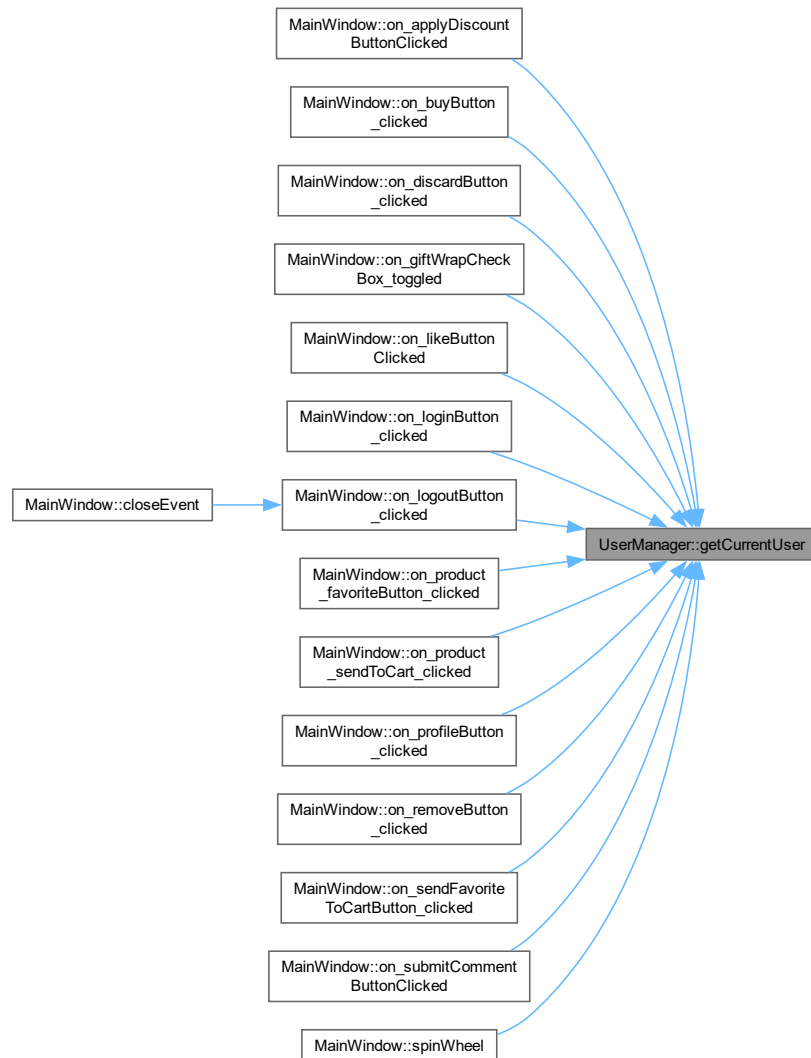
Retrieves the current logged-in user.

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:



### 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**

<i>email</i>	The email address to check.
--------------	-----------------------------

**Returns**

True if the email is already registered, false otherwise.

Searches the user storage file for a matching email.

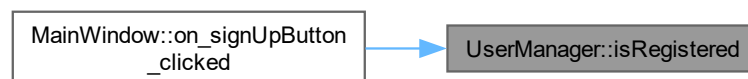
**Parameters**

<i>email</i>	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**

<i>email</i>	The email address provided for login.
<i>password</i>	The password provided for login.
<i>products</i>	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

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

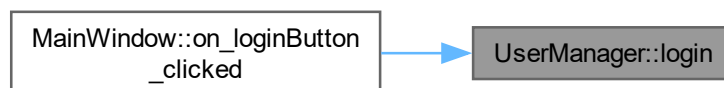
## Returns

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

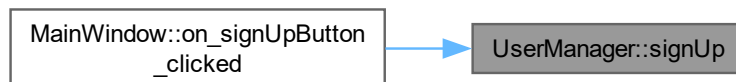
<i>name</i>	The first name of the user.
<i>surname</i>	The last name of the user.
<i>email</i>	The email address of the user.
<i>password</i>	The password for the user.

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

**Parameters**

<i>name</i>	The first name of the user.
<i>surname</i>	The last name of the user.
<i>email</i>	The email address of the user.
<i>password</i>	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](#)



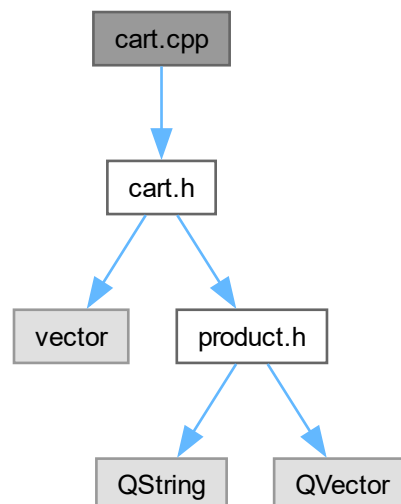
## Chapter 7

# File Documentation

### 7.1 cart.cpp File Reference

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

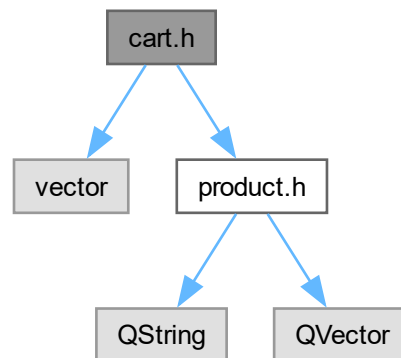
Include dependency graph for cart.cpp:



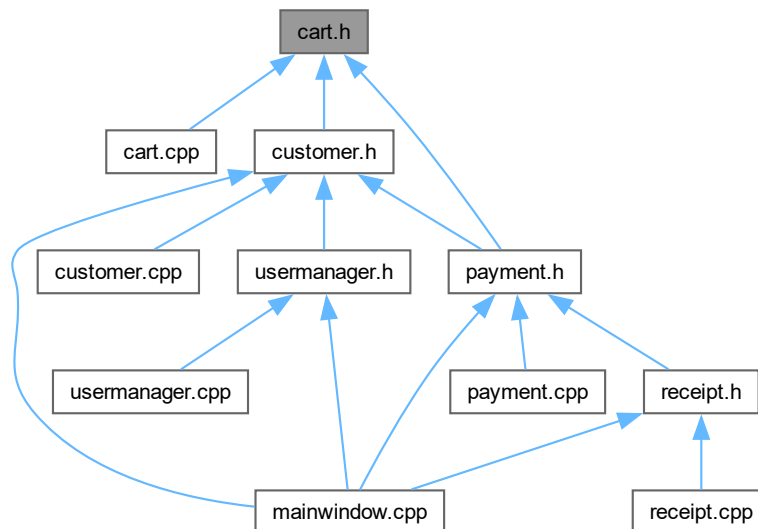
### 7.2 cart.h File Reference

```
#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.](#)

```

00001 #ifndef CART_H
00002 #define CART_H
00003
00004 #include <vector>
00005 #include "product.h"
00006 using namespace std;
00007
00015 class Cart {
00016 private:
00020     vector<Product> products;
00021
00022 public:
00026     Cart();
00027
00033     void addProduct(const Product& product);
00034
00040     void removeProductById(int productId);
00041
00047     const vector<Product>& getProducts() const;
00048
00052     void clearCart();
00053 };
00054
00055 #endif // CART_H

```

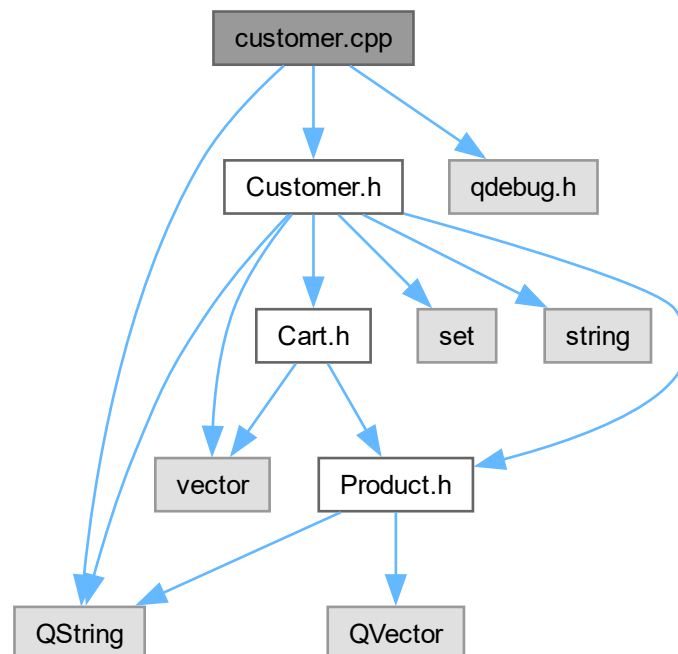
## 7.4 customer.cpp File Reference

```

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

```

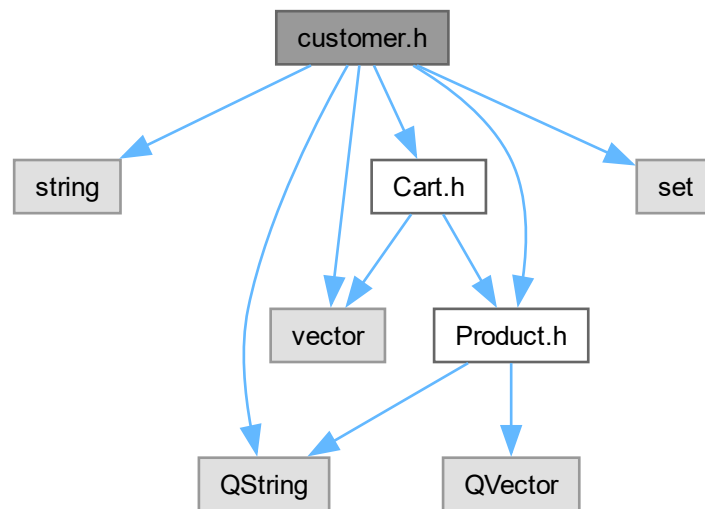
Include dependency graph for customer.cpp:



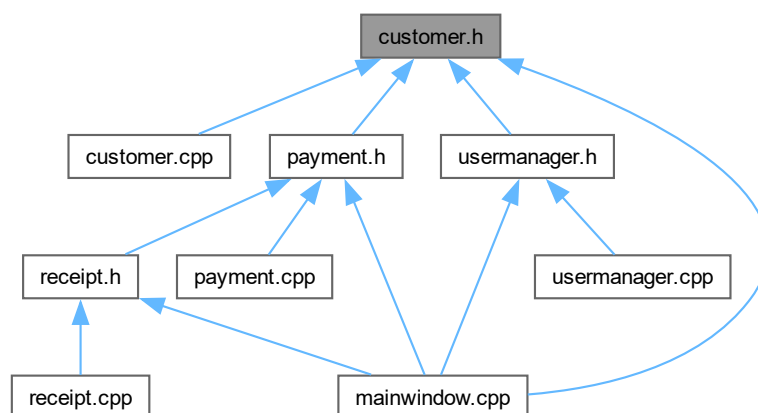
## 7.5 customer.h File Reference

```
#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

<a href="#">Electronics</a>	Represents electronic products.
<a href="#">Clothes</a>	Represents clothing products.

## 7.6 customer.h

[Go to the documentation of this file.](#)

```
00001 // Customer.h
00002 #ifndef CUSTOMER_H
00003 #define CUSTOMER_H
00004
00005 #include <string>
00006 #include <vector>
00007 #include <QString>
00008 #include "Product.h"
00009 #include "Cart.h"
00010 #include <set>
00011
00012 using namespace std;
00013
00014 enum class ProductType {
00015     Electronics,
00016     Clothes
00017 };
00018
00019 class Customer {
00020     std::set<int> likedProductIds;
00021
00022 public:
00023     Cart cart;
00024     Customer(QString name, QString surname, ProductType productType,
00025             QString email, QString password);
00026     ~Customer();
00027
00028     Cart getCartObject();
00029     void addPoint(int amount);
00030     int myPoints() const;
```

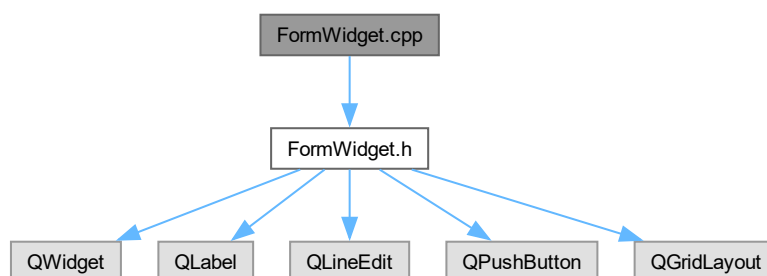
```
00074
00080     bool isCartEmpty(Product prod);
00081
00087     bool addFavorite(Product prod);
00088
00093     void removeFavorite(int id);
00094
00099     QVector<Product> getFavorites();
00100
00106     void setFavorites(QString favorites, QVector<Product> products);
00107
00112     void addCart(Product prod);
00113
00118     void removeCart(int id);
00119
00124     vector<Product> getCart();
00125
00131     bool likeProduct(int productId);
00132
00138     bool hasLikedProduct(int productId) const;
00139
00144     void addPurchasedProduct(const Product& product);
00145
00151     bool hasPurchasedProduct(int productId) const;
00152
00153
00154
00160     QString getName() const {
00161         return name;
00162     }
00163
00164
00170     void setName(const QString& newName) {
00171         name = newName;
00172     }
00173
00174
00180     QString getSurname() const {
00181         return surname;
00182     }
00183
00184
00190     void setSurname(const QString& newSurname) {
00191         surname = newSurname;
00192     }
00193
00194
00200     QString getEmail() const {
00201         return email;
00202     }
00203
00204
00210     void setPassword(const QString& newPassword) {
00211         password = newPassword;
00212     }
00213     //(no getter for security reasons)
00214
00215
00221     ProductType getProductType() const {
00222         return productType;
00223     }
00224
00225
00231     void setProductType(ProductType newProductType) {
00232         productType = newProductType;
00233     }
00234
00235
00241     int getPoints() const {
00242         return point;
00243     }
00244
00245
00251     void setPoints(int newPoints) {
00252         point = newPoints;
00253     }
00254
00255 private:
00259     std::vector<Product> purchasedProducts;
00260
00261     QString name;
00262     QString surname;
00263     QString email;
00264     QString password;
00265     ProductType productType;
00266     int point;
00267
```

```
00268     QVector<Product> favorites;  
00269     QVector<Product> previousOrders;  
00270  
00271 };  
00272  
00273 #endif // CUSTOMER_H
```

## 7.7 FormWidget.cpp File Reference

```
#include "FormWidget.h"
```

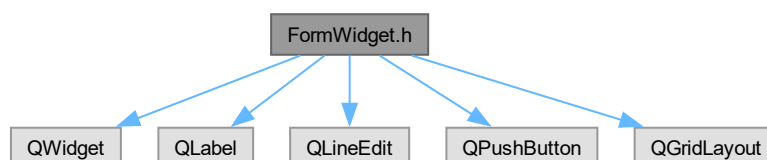
Include dependency graph for FormWidget.cpp:



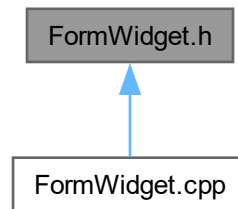
## 7.8 FormWidget.h File Reference

```
#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.](#)

```

00001 #ifndef FORMWIDGET_H
00002 #define FORMWIDGET_H
00003
00004 #include <QWidget>
00005 #include <QLabel>
00006 #include <QLineEdit>
00007 #include <QPushButton>
00008 #include <QGridLayout>
00009
00016 class FormWidget : public QWidget
00017 {
00018     Q_OBJECT
00019
00020 public:
00026     explicit FormWidget(QWidget *parent = nullptr);
00027
00028 private:
00029     QLabel *emailLabel;
00030     QLineEdit *emailField;
00031
00032     QLabel *passwordLabel;
00033     QLineEdit *passwordField;
00034
00035     QPushButton *loginButton;
00036     QPushButton *signupButton;
00037
00038     QGridLayout *formLayout;
00039 };
00040
00041 #endif // FORMWIDGET_H
  
```

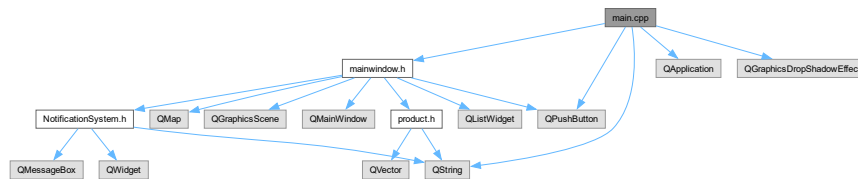
## 7.10 main.cpp File Reference

```

#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

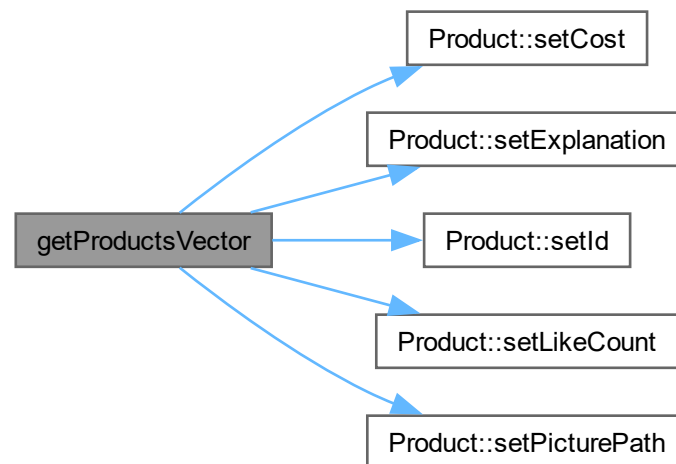
```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "customer.h"
#include "product.h"
#include <QPixmap>
#include <QIcon>
#include <QListWidget>
#include <QLabel>
#include <QDir>
#include <QCoreApplication>
#include <QFile>
#include <QTextStream>
#include <QDebug>
#include <QMessageBox>
#include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



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".

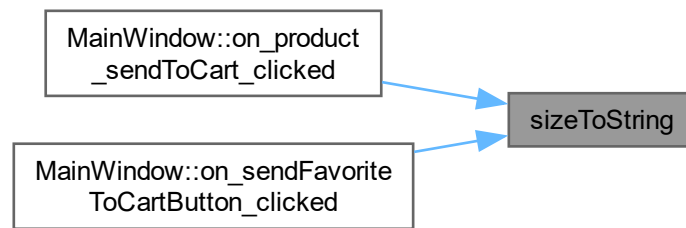
#### Parameters

<code>size</code>	The <code>Product::SIZE</code> 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:



## 7.11.2 Variable Documentation

### 7.11.2.1 userManager

`UserManager` userManager

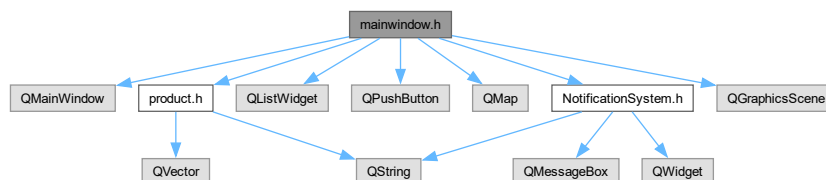
## 7.12 mainwindow.h File Reference

```

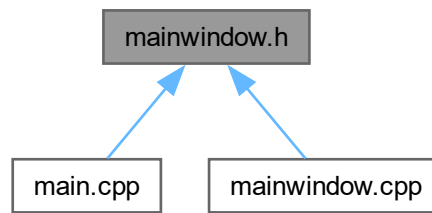
#include <QMainWindow>
#include "product.h"
#include <QListWidget>
#include <QPushButton>
#include <QMap>
#include "NotificationSystem.h"
#include <QGraphicsScene>

```

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.](#)

```

00001
00002 #ifndef MAINWINDOW_H
00003 #define MAINWINDOW_H
00004
00005 #include <QMainWindow>
00006 #include "product.h"
00007 #include <QListWidget>
00008 #include <QPushButton>
00009 #include <QMap>
00010 #include "NotificationSystem.h"
00011 #include <QGraphicsScene>
00012
00013 QT_BEGIN_NAMESPACE
00014 namespace Ui { class MainWindow; }
00015 QT_END_NAMESPACE
00016
00025 class MainWindow : public QMainWindow
00026 {
00027     Q_OBJECT
00028 public:
00034     MainWindow(QWidget *parent = nullptr);
00035
00039     ~MainWindow();
00040
00041 protected:
00050     void closeEvent(QCloseEvent *event) override;
00051
00052 private slots:
00056     void spinWheel();
00057
00063     void applyDiscount(int percentage);
00064
00071     void on_searchButton_clicked();
00072
00078     void on_applyDiscountButtonClicked();
00079

```

```

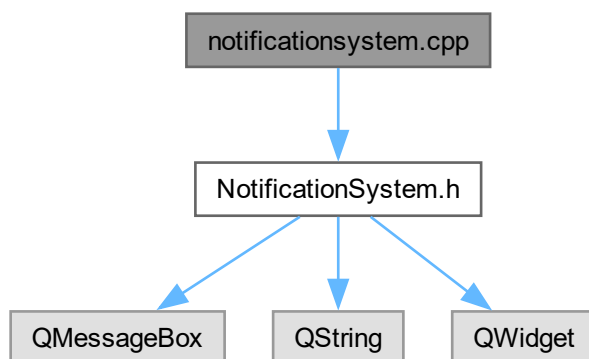
00085     void on_productButtonClicked(Product& product);
00086
00090     void on_productButton_clicked();
00091
00097     void on_likeButtonClicked();
00098
00104     void on_submitCommentButtonClicked();
00105
00113     void displayCommentsForProduct(const Product &product);
00114
00118     void on_gotoSignUp_clicked();
00119
00125     void on_signUpButton_clicked();
00126
00132     void on_loginButton_clicked();
00133
00137     void on_electronicsButton_clicked();
00138
00142     void on_clothesButton_clicked();
00143
00147     void on_mainScreenButton_clicked();
00148
00154     void on_logoutButton_clicked();
00155
00159     void on_product_favoriteButton_clicked();
00160
00164     void on_product_sendToCart_clicked();
00165
00171     void on_profileButton_clicked();
00172
00176     void on_discardButton_clicked();
00177
00181     void on_sendFavoriteToCartButton_clicked();
00182
00186     void on_cartButton_clicked();
00187
00193     void on_buyButton_clicked();
00194
00198     void on_removeButton_clicked();
00199
00205     void on_sizeComboBox_currentTextChanged(const QString &arg1);
00206
00210     void on_pass_to_loginPage_clicked();
00211
00220     void setProductButton(QPushButton *button, const QString &picturePath);
00221
00225     void setupProductButtons();
00226
00230     void showDiscountColors();
00231
00239     void on_giftWrapCheckBox_toggled(bool checked);
00240
00241 private:
00242     Ui::MainWindow *ui;
00243     QListWidget *commentListWidget;
00244     NotificationSystem *notificationSystem;
00245     QVector<Product> products;
00246     QVector<QPushButton*> productButtons;
00247     QMap<QPushButton*, Product*> buttonProductMap;
00248     Product* currentProduct;
00249
00251     QGraphicsScene *scene;
00252     QGraphicsEllipseItem *wheel;
00253     QTimer *timer;
00254     int currentAngle;
00255     int targetAngle;
00256     QStringList wheelRewards;
00257     double currentSpeed;
00258     double minSpeed;
00259     double deceleration;
00260     QList<QGraphicsPathItem*> slices;
00261     QMap<int, QString> colorMap;
00262     QMap<QString, int> discountMap;
00263     QTimer *blinkTimer;
00264     int blinkCount;
00265     bool blinkState;
00266     int chosenSliceIndex;
00267     int currentDiscount;
00268     bool discountApplied;
00269
00271     const double giftWrapFee = 10.0;
00272
00274     void connectProductButtons();
00275 };
00276
00277 #endif // MAINWINDOW_H

```

## 7.14 notificationsystem.cpp File Reference

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

Include dependency graph for notificationsystem.cpp:



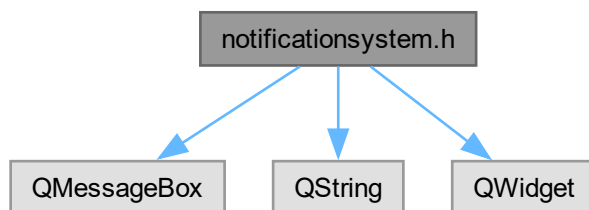
## 7.15 notificationsystem.h File Reference

```
#include <QMessageBox>
```

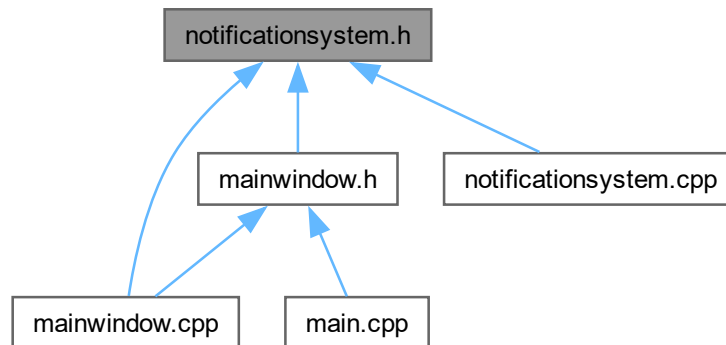
```
#include <QString>
```

```
#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 notificationssystem.h

[Go to the documentation of this file.](#)

```

00001 #ifndef NOTIFICATIONSYSTEM_H
00002 #define NOTIFICATIONSYSTEM_H
00003
00004 #include <QMessageBox>
00005 #include <QString>
00006 #include <QWidget>
00007
00012 class NotificationSystem {
00013 public:
00019     explicit NotificationSystem(QWidget *parent = nullptr);
00020
00027     void showInfo(const QString &title, const QString &message);
00028
00035     void showWarning(const QString &title, const QString &message);
00036
00043     void showError(const QString &title, const QString &message);
00044
00045 private:
00046     QWidget *parentWidget;
00047 };
00048
00049 #endif // NOTIFICATIONSYSTEM_H
  
```

## 7.17 payment.cpp File Reference

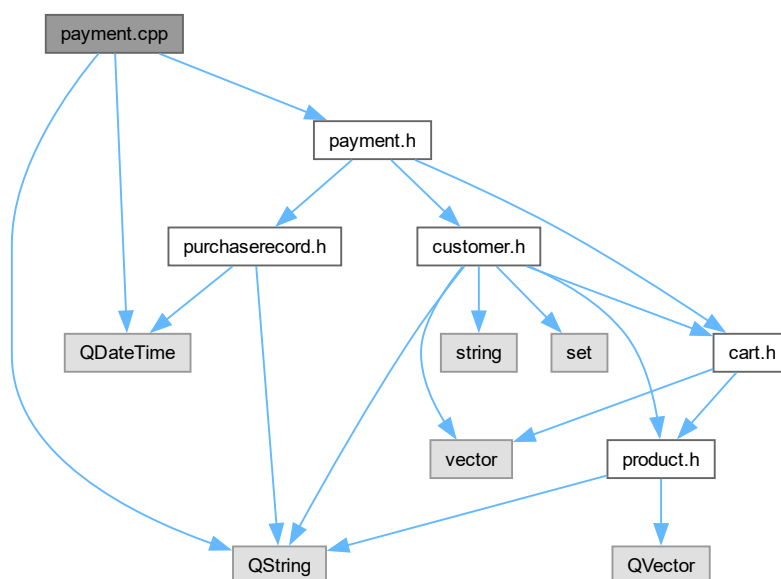
```

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



```
#include <QString>
```

Include dependency graph for payment.cpp:



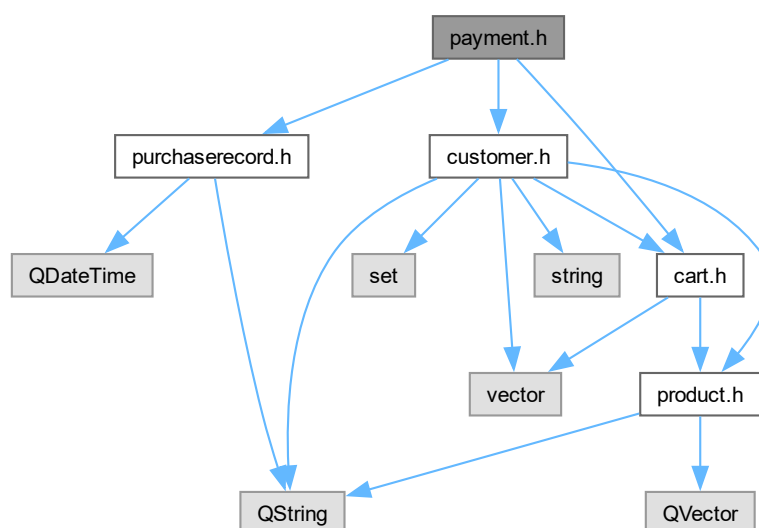
## 7.18 payment.h File Reference

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

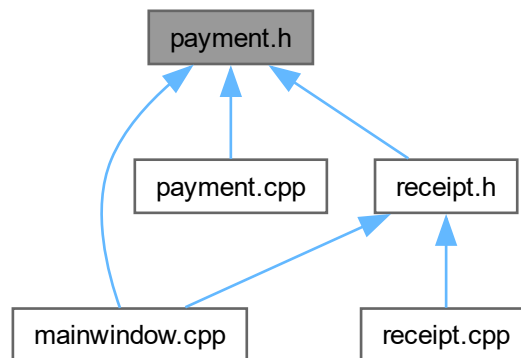
```
#include "cart.h"
```

```
#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

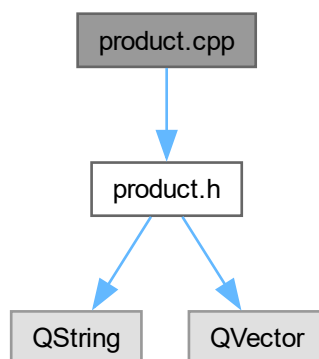
[Go to the documentation of this file.](#)

```
00001 #ifndef PAYMENT_H
00002 #define PAYMENT_H
00003
00004 #include "purchaserecord.h"
00005 #include "cart.h"
00006 #include "customer.h"
00007
00011 enum Discount {
00012     NoDiscount,
00013     D10,
00014     D20,
00015     D50
00016 };
00017
00024 class Payment : public PurchaseRecord {
00025 private:
00026     Customer customer;
00027     Cart cart;
00028     Discount discount;
00029
00030 public:
00038     Payment(Customer customer, Cart cart, Discount discount);
00039
00045     void setDiscount(Discount discount);
00046
00052     Discount getDiscount() const;
00053
00059     void setCustomer(const Customer& customer);
00060
00066     Customer getCustomer() const;
00067
00073     void setCart(const Cart& cart);
00074
00080     Cart getCart() const;
00081
00087     double grandTotal() const;
00088
00094     double total() const;
00095
00101     double applyDiscount() const;
00102
00108     std::string discountPercentage() const;
00109
00115     void applyDiscountCode(const QString &code);
00116
00122     QString getRecordDetails() const override;
00123 };
00124
00125 #endif // PAYMENT_H
```

## 7.20 product.cpp File Reference

```
#include "product.h"
```

Include dependency graph for product.cpp:

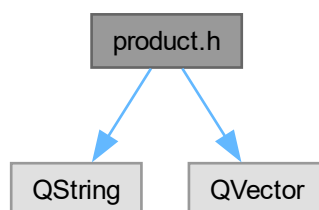


## 7.21 product.h File Reference

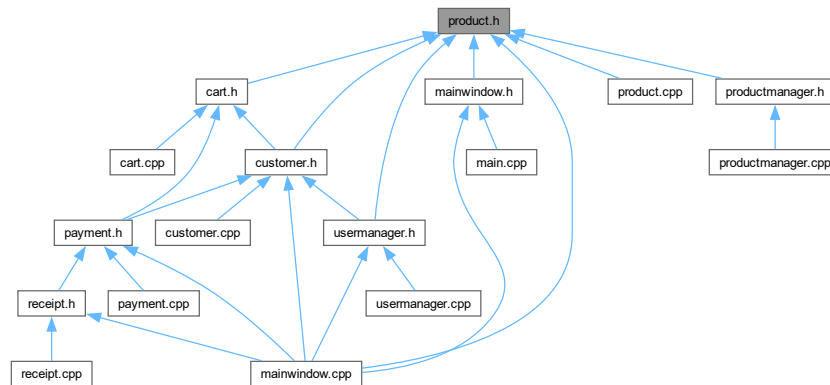
```
#include <QString>
```

```
#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.](#)

```

00001 #ifndef PRODUCT_H
00002 #define PRODUCT_H
00003
00004 #include <QString>
00005 #include <QVector>
00006
00013 class Product {
00014
00015 private:
00016     int id;
00017     QString picturePath;
00018     QString explanation;
00019     double cost;
00020     int likeCount;
00021     QVector<QString> comments;
00022
00023 public:
00027     enum class SIZE {
00028         XSMALL,
00029         SMALL,
00030         MEDIUM,
00031         LARGE,
00032         XLARGE
00033     };
00034
00035     SIZE selectedSize;
00036
00040     Product();
00041
00051     Product(int id, QString picturePath, QString explanation, double cost, int likeCount);
00052
00058     Product(const Product& temp);
00059
00065     int getId() const;
00066
00072     void setId(int value);
00073
00079     QString getPicturePath() const;
00080
00086     void setPicturePath(const QString& value);
00087

```

```

00093     QString getExplanation() const;
00094
00100     void setExplanation(const QString& value);
00101
00107     double getCost() const;
00108
00114     void setCost(double value);
00115
00121     int getLikeCount() const;
00122
00128     void setLikeCount(int value);
00129
00135     QVector<QString> getComments() const;
00136
00142     void setComments(const QVector<QString>& value);
00143
00149     void addComment(const QString &comment);
00150
00154     void likeProduct();
00155
00161     void unlikeProduct();
00162
00168     SIZE getSelectedSize() const;
00169
00175     void setSelectedSize(SIZE size);
00176
00182     QString getSizeString() const;
00183
00189     QString toQString() const;
00190 };
00191
00192 #endif // PRODUCT_H

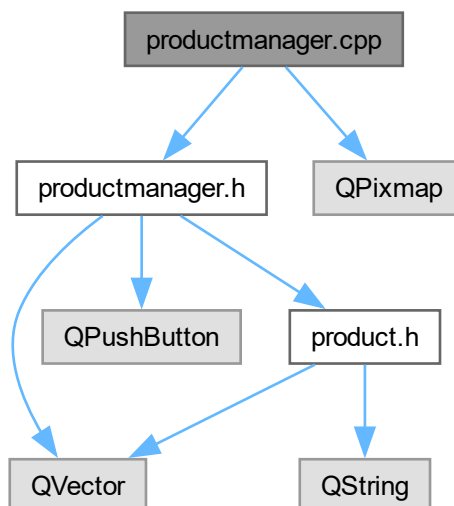
```

## 7.23 productmanager.cpp File Reference

```
#include "productmanager.h"
```

```
#include <QPixmap>
```

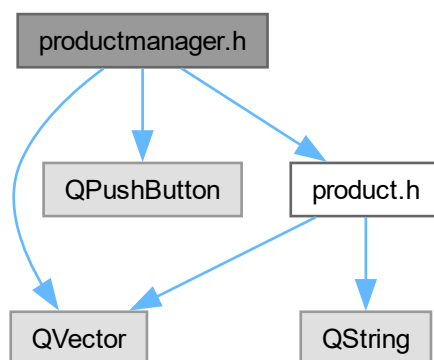
Include dependency graph for productmanager.cpp:



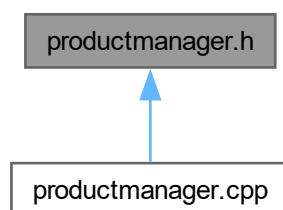
## 7.24 productmanager.h File Reference

```
#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.](#)

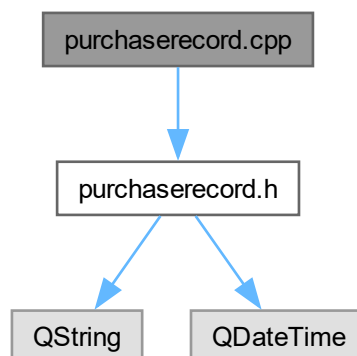
```
00001 #ifndef PRODUCTMANAGER_H
```

```
00002 #define PRODUCTMANAGER_H
00003
00004 #include <QVector>
00005 #include <QPushButton>
00006 #include "product.h"
00007
00008
00009
00016 class ProductManager {
00017 public:
00023     ProductManager(const QVector<Product> &products);
00024
00034     void setupProductButtons(const QVector<QPushButton *> &buttons, int startIndex = 0);
00035
00045     Product* getProductByIndex(int index);
00046
00047 private:
00048     QVector<Product> products;
00049 };
00050
00051 #endif // PRODUCTMANAGER_H
```

## 7.26 purchaserecord.cpp File Reference

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

Include dependency graph for purchaserecord.cpp:

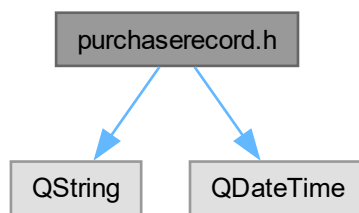


## 7.27 purchaserecord.h File Reference

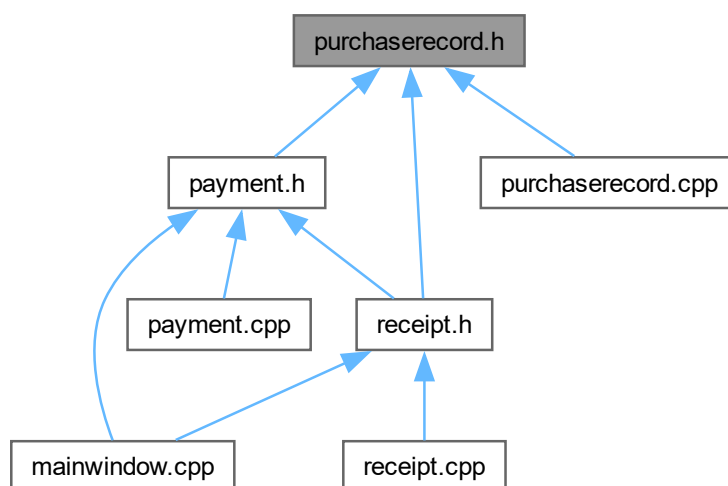
```
#include <QString>
#include <QDateTime>
```



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.](#)

```
00001 #ifndef PURCHASERECORD_H
00002 #define PURCHASERECORD_H
00003
00004 #include <QString>
```

```

00005 #include <QDateTime>
00006
00014 class PurchaseRecord {
00015 protected:
00016     QDateTime purchaseDate;
00017     double totalAmount;
00018
00019 public:
00025     PurchaseRecord();
00026
00033     PurchaseRecord(QDateTime date, double amount);
00034
00042     virtual QString getRecordDetails() const = 0;
00043
00049     QDateTime getPurchaseDate() const;
00050
00056     void setPurchaseDate(const QDateTime& date);
00057
00063     double getTotalAmount() const;
00064
00070     void setTotalAmount(double amount);
00071
00077     virtual ~PurchaseRecord() = default;
00078 };
00079
00080 #endif // PURCHASERECORD_H

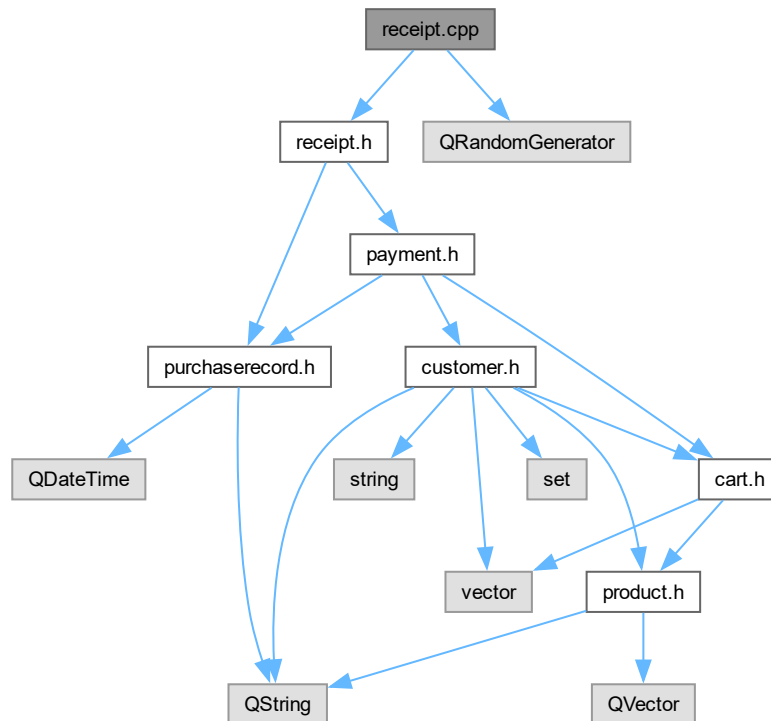
```

## 7.29 receipt.cpp File Reference

```

#include "receipt.h"
#include <QRandomGenerator>
Include dependency graph for receipt.cpp:

```

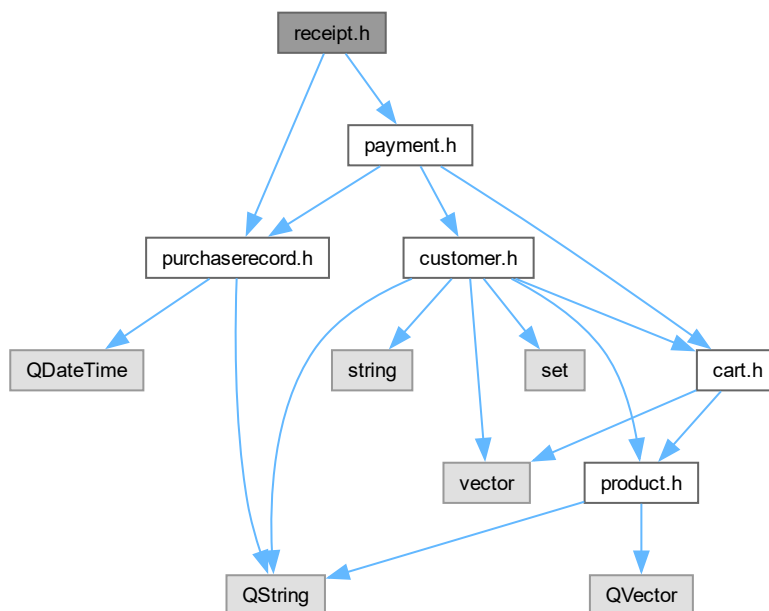


## 7.30 receipt.h File Reference

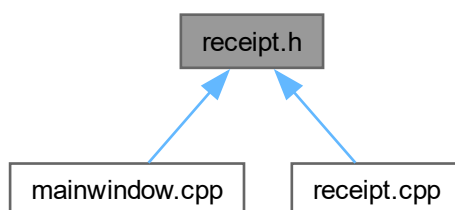
```
#include "purchaserecord.h"
```

```
#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.](#)

```

00001 #ifndef RECEIPT_H
00002 #define RECEIPT_H
00003
00004 #include "purchaserecord.h"
00005 #include "payment.h"
00006
00014 class Receipt : public PurchaseRecord {
00015     Payment payment;
00016
00017 public:
00023     Receipt(const Payment& payment);
00024
00032     int addPoint();
00033
00041     QString orderNo() const;
00042
00051     QString toString() const;
00052
00060     QString getRecordDetails() const override;
00061 };
00062
00063 #endif // RECEIPT_H

```

## 7.32 usermanager.cpp File Reference

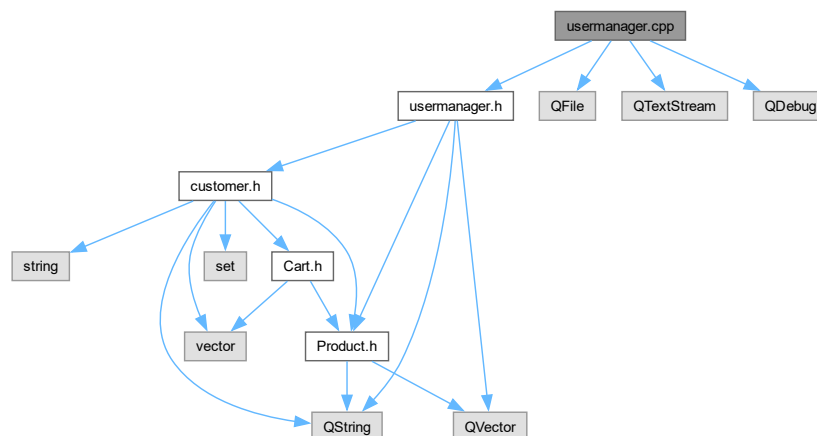
Implementation of the [UserManager](#) class.

```

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

```

Include dependency graph for usermanager.cpp:



### 7.32.1 Detailed Description

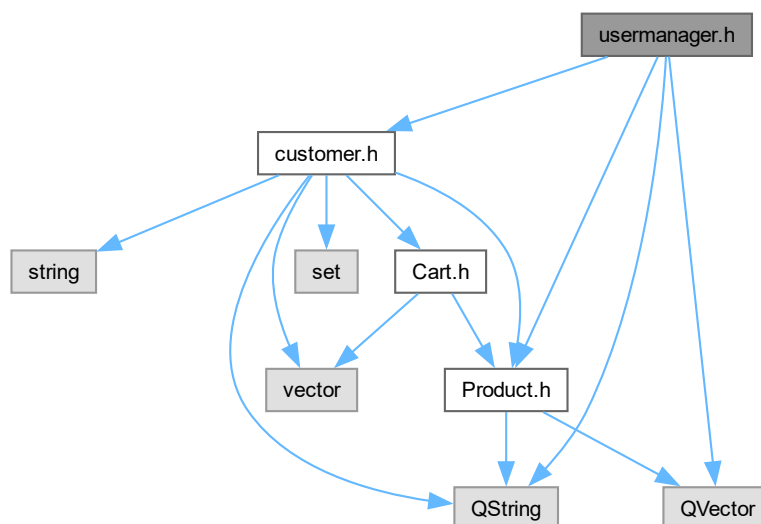
Implementation of the [UserManager](#) class.

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

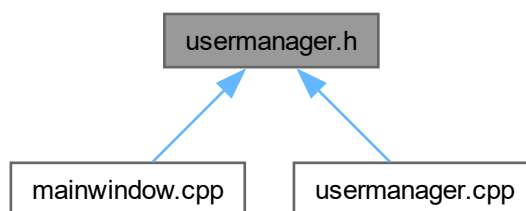
## 7.33 usermanager.h File Reference

```
#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.](#)

```
00001 #ifndef USERMANAGER_H
00002 #define USERMANAGER_H
00003
00004 #include "customer.h"
00005 #include "product.h"
00006 #include <QString>
00007 #include <QVector>
00008
00015 class UserManager {
00016 private:
00017     Customer currentUser;
00018
00019 public:
00023     UserManager();
00024
00030     Customer& getCurrentUser();
00031
00043     bool login(const QString &email, const QString &password, const QVector<Product> &products);
00044
00053     bool isRegistered(const QString &email);
00054
00066     void signUp(const QString &name, const QString &surname, const QString &email, const QString
&password);
00067 };
00068
00069 #endif // USERMANAGER_H
```

# Index

- ~Customer
  - Customer, [16](#)
- ~MainWindow
  - MainWindow, [36](#)
- ~PurchaseRecord
  - PurchaseRecord, [89](#)
- addCart
  - Customer, [17](#)
- addComment
  - Product, [75](#)
- addFavorite
  - Customer, [17](#)
- addPoint
  - Customer, [18](#)
  - Receipt, [94](#)
- addProduct
  - Cart, [12](#)
- addPurchasedProduct
  - Customer, [19](#)
- applyDiscount
  - MainWindow, [37](#)
  - Payment, [65](#)
- applyDiscountCode
  - Payment, [66](#)
- blinkCount
  - MainWindow, [57](#)
- blinkState
  - MainWindow, [57](#)
- blinkTimer
  - MainWindow, [57](#)
- buttonProductMap
  - MainWindow, [57](#)
- Cart, [11](#)
  - addProduct, [12](#)
  - Cart, [12](#)
  - clearCart, [12](#)
  - getProducts, [12](#)
  - products, [14](#)
  - removeProductById, [13](#)
- cart
  - Customer, [28](#)
  - Payment, [70](#)
- cart.cpp, [103](#)
- cart.h, [103](#)
- chosenSliceIndex
  - MainWindow, [57](#)
- clearCart
  - Cart, [12](#)
- closeEvent
  - MainWindow, [37](#)
- Clothes
  - customer.h, [107](#)
- colorMap
  - MainWindow, [57](#)
- commentListWidget
  - MainWindow, [57](#)
- comments
  - Product, [84](#)
- connectProductButtons
  - MainWindow, [38](#)
- cost
  - Product, [84](#)
- currentAngle
  - MainWindow, [57](#)
- currentDiscount
  - MainWindow, [57](#)
- currentProduct
  - MainWindow, [57](#)
- currentSpeed
  - MainWindow, [58](#)
- currentUser
  - UserManager, [102](#)
- Customer, [14](#)
  - ~Customer, [16](#)
  - addCart, [17](#)
  - addFavorite, [17](#)
  - addPoint, [18](#)
  - addPurchasedProduct, [19](#)
  - cart, [28](#)
  - Customer, [16](#)
  - email, [28](#)
  - favorites, [29](#)
  - getCart, [19](#)
  - getCartObject, [20](#)
  - getEmail, [21](#)
  - getFavorites, [21](#)
  - getName, [21](#)
  - getPoints, [22](#)
  - getProductType, [22](#)
  - getSurname, [22](#)
  - hasLikedProduct, [23](#)
  - hasPurchasedProduct, [23](#)
  - isCartEmpty, [24](#)
  - likedProductIds, [29](#)
  - likeProduct, [24](#)
  - myPoints, [25](#)

- name, 29
- password, 29
- point, 29
- previousOrders, 29
- productType, 29
- purchasedProducts, 29
- removeCart, 25
- removeFavorite, 26
- setFavorites, 26
- setName, 27
- setPassword, 27
- setPoints, 28
- setProductType, 28
- setSurname, 28
- surname, 30
- customer
  - Payment, 70
- customer.cpp, 105
- customer.h, 106
  - Clothes, 107
  - Electronics, 107
  - ProductType, 107
- D10
  - payment.h, 120
- D20
  - payment.h, 120
- D50
  - payment.h, 120
- deceleration
  - MainWindow, 58
- Discount
  - payment.h, 120
- discount
  - Payment, 70
- discountApplied
  - MainWindow, 58
- discountMap
  - MainWindow, 58
- discountPercentage
  - Payment, 66
- displayCommentsForProduct
  - MainWindow, 38
- Electronics
  - customer.h, 107
- email
  - Customer, 28
- emailField
  - FormWidget, 32
- emailLabel
  - FormWidget, 32
- explanation
  - Product, 84
- favorites
  - Customer, 29
- formLayout
  - FormWidget, 32
- FormWidget, 30
  - emailField, 32
  - emailLabel, 32
  - formLayout, 32
  - FormWidget, 31
  - loginButton, 32
  - passwordField, 32
  - passwordLabel, 32
  - signupButton, 32
- FormWidget.cpp, 109
- FormWidget.h, 109
- getCart
  - Customer, 19
  - Payment, 66
- getCartObject
  - Customer, 20
- getComments
  - Product, 75
- getCost
  - Product, 75
- getCurrentUser
  - UserManager, 98
- getCustomer
  - Payment, 67
- getDiscount
  - Payment, 67
- getEmail
  - Customer, 21
- getExplanation
  - Product, 76
- getFavorites
  - Customer, 21
- getId
  - Product, 76
- getLikeCount
  - Product, 77
- getName
  - Customer, 21
- getPicturePath
  - Product, 77
- getPoints
  - Customer, 22
- getProductByIndex
  - ProductManager, 86
- getProducts
  - Cart, 12
- getProductsVector
  - mainwindow.cpp, 112
- getProductType
  - Customer, 22
- getPurchaseDate
  - PurchaseRecord, 89
- getRecordDetails
  - Payment, 67
  - PurchaseRecord, 89
  - Receipt, 94
- getSelectedSize
  - Product, 78



- getSizeString
  - Product, 78
- getSurname
  - Customer, 22
- getTotalAmount
  - PurchaseRecord, 90
- giftWrapFee
  - MainWindow, 58
- grandTotal
  - Payment, 68
- hasLikedProduct
  - Customer, 23
- hasPurchasedProduct
  - Customer, 23
- id
  - Product, 84
- isCartEmpty
  - Customer, 24
- isRegistered
  - UserManager, 99
- LARGE
  - Product, 74
- likeCount
  - Product, 84
- likedProductIds
  - Customer, 29
- likeProduct
  - Customer, 24
  - Product, 79
- login
  - UserManager, 100
- loginButton
  - FormWidget, 32
- main
  - main.cpp, 111
- main.cpp, 110
  - main, 111
- MainWindow, 33
  - ~MainWindow, 36
  - applyDiscount, 37
  - blinkCount, 57
  - blinkState, 57
  - blinkTimer, 57
  - buttonProductMap, 57
  - chosenSliceIndex, 57
  - closeEvent, 37
  - colorMap, 57
  - commentListWidget, 57
  - connectProductButtons, 38
  - currentAngle, 57
  - currentDiscount, 57
  - currentProduct, 57
  - currentSpeed, 58
  - deceleration, 58
  - discountApplied, 58
  - discountMap, 58
  - displayCommentsForProduct, 38
  - giftWrapFee, 58
  - MainWindow, 36
  - minSpeed, 58
  - notificationSystem, 58
  - on\_applyDiscountButtonClicked, 39
  - on\_buyButton\_clicked, 40
  - on\_cartButton\_clicked, 41
  - on\_clothesButton\_clicked, 41
  - on\_discardButton\_clicked, 41
  - on\_electronicsButton\_clicked, 42
  - on\_giftWrapCheckBox\_toggled, 42
  - on\_gotoSignUp\_clicked, 43
  - on\_likeButtonClicked, 43
  - on\_loginButton\_clicked, 44
  - on\_logoutButton\_clicked, 45
  - on\_mainScreenButton\_clicked, 45
  - on\_pass\_to\_loginPage\_clicked, 46
  - on\_product\_favoriteButton\_clicked, 46
  - on\_product\_sendToCart\_clicked, 46
  - on\_productButton\_clicked, 47
  - on\_productButtonClicked, 48
  - on\_profileButton\_clicked, 49
  - on\_removeButton\_clicked, 50
  - on\_searchButton\_clicked, 51
  - on\_sendFavoriteToCartButton\_clicked, 51
  - on\_signUpButton\_clicked, 52
  - on\_sizeComboBox\_currentTextChanged, 53
  - on\_submitCommentButtonClicked, 54
  - productButtons, 58
  - products, 58
  - scene, 59
  - setProductButton, 54
  - setupProductButtons, 55
  - showDiscountColors, 56
  - slices, 59
  - spinWheel, 56
  - targetAngle, 59
  - timer, 59
  - ui, 59
  - wheel, 59
  - wheelRewards, 59
- mainwindow.cpp, 111
  - getProductsVector, 112
  - sizeToString, 113
  - userManager, 114
- mainwindow.h, 114
- MEDIUM
  - Product, 74
- minSpeed
  - MainWindow, 58
- myPoints
  - Customer, 25
- name
  - Customer, 29
- NoDiscount
  - payment.h, 120

- NotificationSystem, 60
  - NotificationSystem, 60
  - parentWidget, 62
  - showError, 60
  - showInfo, 61
  - showWarning, 61
- notificationSystem
  - MainWindow, 58
- notificationsystem.cpp, 117
- notificationsystem.h, 117
- on\_applyDiscountButtonClicked
  - MainWindow, 39
- on\_buyButton\_clicked
  - MainWindow, 40
- on\_cartButton\_clicked
  - MainWindow, 41
- on\_clothesButton\_clicked
  - MainWindow, 41
- on\_discardButton\_clicked
  - MainWindow, 41
- on\_electronicsButton\_clicked
  - MainWindow, 42
- on\_giftWrapCheckBox\_toggled
  - MainWindow, 42
- on\_gotoSignUp\_clicked
  - MainWindow, 43
- on\_likeButtonClicked
  - MainWindow, 43
- on\_loginButton\_clicked
  - MainWindow, 44
- on\_logoutButton\_clicked
  - MainWindow, 45
- on\_mainScreenButton\_clicked
  - MainWindow, 45
- on\_pass\_to\_loginPage\_clicked
  - MainWindow, 46
- on\_product\_favoriteButton\_clicked
  - MainWindow, 46
- on\_product\_sendToCart\_clicked
  - MainWindow, 46
- on\_productButton\_clicked
  - MainWindow, 47
- on\_productButtonClicked
  - MainWindow, 48
- on\_profileButton\_clicked
  - MainWindow, 49
- on\_removeButton\_clicked
  - MainWindow, 50
- on\_searchButton\_clicked
  - MainWindow, 51
- on\_sendFavoriteToCartButton\_clicked
  - MainWindow, 51
- on\_signUpButton\_clicked
  - MainWindow, 52
- on\_sizeComboBox\_currentTextChanged
  - MainWindow, 53
- on\_submitCommentButtonClicked
  - MainWindow, 54
- orderNo
  - Receipt, 95
- parentWidget
  - NotificationSystem, 62
- password
  - Customer, 29
- passwordField
  - FormWidget, 32
- passwordLabel
  - FormWidget, 32
- Payment, 63
  - applyDiscount, 65
  - applyDiscountCode, 66
  - cart, 70
  - customer, 70
  - discount, 70
  - discountPercentage, 66
  - getCart, 66
  - getCustomer, 67
  - getDiscount, 67
  - getRecordDetails, 67
  - grandTotal, 68
  - Payment, 65
  - setCart, 68
  - setCustomer, 69
  - setDiscount, 69
  - total, 69
- payment
  - Receipt, 96
- payment.cpp, 118
- payment.h, 119
  - D10, 120
  - D20, 120
  - D50, 120
  - Discount, 120
  - NoDiscount, 120
- picturePath
  - Product, 84
- point
  - Customer, 29
- previousOrders
  - Customer, 29
- Product, 71
  - addComment, 75
  - comments, 84
  - cost, 84
  - explanation, 84
  - getComments, 75
  - getCost, 75
  - getExplanation, 76
  - getId, 76
  - getLikeCount, 77
  - getPicturePath, 77
  - getSelectedSize, 78
  - getSizeString, 78
  - id, 84
  - LARGE, 74
  - likeCount, 84

- likeProduct, 79
- MEDIUM, 74
- picturePath, 84
- Product, 74
- selectedSize, 84
- setComments, 79
- setCost, 80
- setExplanation, 80
- setId, 80
- setLikeCount, 82
- setPicturePath, 82
- setSelectedSize, 83
- SIZE, 73
- SMALL, 74
- toQString, 83
- unlikeProduct, 83
- XLARGE, 74
- XSMALL, 74
- product.cpp, 122
- product.h, 122
- productButtons
  - MainWindow, 58
- ProductManager, 85
  - getProductByIndex, 86
  - ProductManager, 85
  - products, 87
  - setupProductButtons, 86
- productmanager.cpp, 124
- productmanager.h, 125
- products
  - Cart, 14
  - MainWindow, 58
  - ProductManager, 87
- ProductType
  - customer.h, 107
- productType
  - Customer, 29
- purchaseDate
  - PurchaseRecord, 91
- purchasedProducts
  - Customer, 29
- PurchaseRecord, 88
  - ~PurchaseRecord, 89
  - getPurchaseDate, 89
  - getRecordDetails, 89
  - getTotalAmount, 90
  - purchaseDate, 91
  - PurchaseRecord, 89
  - setPurchaseDate, 90
  - setTotalAmount, 90
  - totalAmount, 91
- purchaserecord.cpp, 126
- purchaserecord.h, 126
- Receipt, 91
  - addPoint, 94
  - getRecordDetails, 94
  - orderNo, 95
  - payment, 96
  - Receipt, 93
  - toString, 95
- receipt.cpp, 128
- receipt.h, 129
- removeCart
  - Customer, 25
- removeFavorite
  - Customer, 26
- removeProductById
  - Cart, 13
- scene
  - MainWindow, 59
- selectedSize
  - Product, 84
- setCart
  - Payment, 68
- setComments
  - Product, 79
- setCost
  - Product, 80
- setCustomer
  - Payment, 69
- setDiscount
  - Payment, 69
- setExplanation
  - Product, 80
- setFavorites
  - Customer, 26
- setId
  - Product, 80
- setLikeCount
  - Product, 82
- setName
  - Customer, 27
- setPassword
  - Customer, 27
- setPicturePath
  - Product, 82
- setPoints
  - Customer, 28
- setProductButton
  - MainWindow, 54
- setProductType
  - Customer, 28
- setPurchaseDate
  - PurchaseRecord, 90
- setSelectedSize
  - Product, 83
- setSurname
  - Customer, 28
- setTotalAmount
  - PurchaseRecord, 90
- setupProductButtons
  - MainWindow, 55
  - ProductManager, 86
- showDiscountColors
  - MainWindow, 56
- showError

- NotificationSystem, [60](#)
- showInfo
  - NotificationSystem, [61](#)
- showWarning
  - NotificationSystem, [61](#)
- signUp
  - UserManager, [101](#)
- signupButton
  - FormWidget, [32](#)
- SIZE
  - Product, [73](#)
- sizeToString
  - mainwindow.cpp, [113](#)
- slices
  - MainWindow, [59](#)
- SMALL
  - Product, [74](#)
- spinWheel
  - MainWindow, [56](#)
- surname
  - Customer, [30](#)
- targetAngle
  - MainWindow, [59](#)
- timer
  - MainWindow, [59](#)
- toQString
  - Product, [83](#)
- toString
  - Receipt, [95](#)
- total
  - Payment, [69](#)
- totalAmount
  - PurchaseRecord, [91](#)
- Ui, [9](#)
- ui
  - MainWindow, [59](#)
- unlikeProduct
  - Product, [83](#)
- UserManager, [97](#)
  - currentUser, [102](#)
  - getCurrentUser, [98](#)
  - isRegistered, [99](#)
  - login, [100](#)
  - signUp, [101](#)
  - UserManager, [98](#)
- userManager
  - mainwindow.cpp, [114](#)
- userManager.cpp, [130](#)
- userManager.h, [131](#)
- wheel
  - MainWindow, [59](#)
- wheelRewards
  - MainWindow, [59](#)
- XLARGE
  - Product, [74](#)
- XSMALL
  - Product, [74](#)