Warehouse Simulator

1.0.0

Generated by Doxygen 1.10.0

Chapter 1

Warehouse-simulator

The Warehouse Simulation project is a warehouse management support system that allows you to simulate and optimize warehouse processes. Thanks to it, users can model various scenarios, analyze results and make decisions in real time. The project integrates simulation algorithms with an interactive user interface, which allows for dynamic monitoring of warehouse stock and quick response to changes in the business environment.

The main features of the project include defining warehouses, products, attributes, simulating warehouse operation in a time loop, generating events, user intervention and generating reports with simulation results. The aim of the project is to provide a tool supporting effective warehouse management, which will allow companies to increase operational efficiency and maximize profits.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 Ui Namespace Reference

Chapter 7

Class Documentation

7.1 Event Class Reference

The Event class represents an event in the warehouse simulation.

`#include <Event.h>`

Public Member Functions

- Event (QString eventType, QDateTime time)
  Constructor for the Event class.
- QString getEventType () const
  Retrieves the type of the event.
- QDateTime getTime () const
  Retrieves the timestamp of the event.

Static Public Member Functions

- static Event generateEvent (QString eventType, int seed)
  Generates a random event based on the given seed.

7.1.1 Detailed Description

The Event class represents an event in the warehouse simulation.

This class encapsulates the details of an event, including its type and the time it occurred.

Definition at line 17 of file Event.h.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 Event()

```
Event::Event (
            QString eventType,
            QDateTime time )
```

Constructor for the Event class.

Construct a new Event object.

Initializes a new instance of the Event class with the specified type and time.

Parameters

eventType	The type of the event.
time	The timestamp when the event occurred.

Definition at line 12 of file Event.cpp.

7.1.3 Member Function Documentation

7.1.3.1 generateEvent()

```
Event Event::generateEvent (
            QString eventType,
            int seed )  [static]
```

Generates a random event based on the given seed.

Generates an event with a random time offset.

This static method creates a new event with a random time offset based on the provided seed.

Parameters

eventType	The type of the event to generate.
seed	The seed for the random number generator.

Returns

A new Event instance with the specified type and a random time.

Parameters

eventType	The type of the event to generate.
seed	The seed for the random number generator.

Returns

A new Event instance with the specified type and a random time.

Definition at line 25 of file Event.cpp.

7.1.3.2 getEventType()

```
QString Event::getEventType ( ) const
```

Retrieves the type of the event.

Returns

The event type as a QString.

Definition at line 38 of file Event.cpp.

7.1.3.3 getTime()

```
QDateTime Event::getTime ( ) const
```

Retrieves the timestamp of the event.

Returns

The event time as a QDateTime.

Definition at line 48 of file Event.cpp.

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

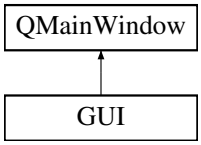
- src/Event/Event.h
- src/Event/Event.cpp

7.2 GUI Class Reference

The GUI class inherits from QMainWindow and represents the main window of the application.

```
#include <gui.h>
```

Inheritance diagram for GUI:



Public Member Functions

- GUI (GUIElement *parent=nullptr)
  Constructor for the GUI class.
- ~GUI ()
  Destructor for the GUI class.
- void render ()
  Renders the graphical user interface elements.

7.2.1 Detailed Description

The GUI class inherits from QMainWindow and represents the main window of the application.

This class manages the user interface for the application. It sets up the main window and its associated widgets, and handles user interactions.

Definition at line 73 of file gui.h.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 GUI()

```
GUI::GUI (
            GUIElement * parent = nullptr )  [explicit]
```

Constructor for the GUI class.

Parameters

parent	Pointer to the parent widget, if any.

Definition at line 14 of file gui.cpp.

7.2.2.2 ~GUI()

```
GUI::~GUI ( )
```

Destructor for the GUI class.

Definition at line 25 of file gui.cpp.

7.2.3 Member Function Documentation

7.2.3.1 render()

```
void GUI::render ( )
```

Renders the graphical user interface elements.

This function is responsible for rendering the GUI elements on the screen. It ensures that all widgets (buttons, labels, etc.) are displayed correctly based on their current state and properties.

Definition at line 33 of file gui.cpp.

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

- src/gui/gui.h
- src/gui/gui.cpp

7.3 Product Class Reference

The Product class represents a product with a name, price, and quantity.

```
#include <Product.h>
```

Public Member Functions

- Product (int productId, QString name, double price, int quantity)
  Construct a new Product object.
- status sell (int quantityToSell)
  Sell a quantity of the product.
- status updatePrice (double newPrice)
  Update the price of the product.
- status changeQuantity (int quantity)
  Change the quantity of the product.
- QString getName () const
  Get the name of the product.
- double getPrice () const
  Get the price of the product.
- int getQuantity () const
  Get the quantity of the product.

Public Attributes

- int productId

  ID of the product.

7.3.1 Detailed Description

The Product class represents a product with a name, price, and quantity.

This class provides methods to sell the product, update its price, and change its quantity.

Definition at line 20 of file Product.h.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 Product()

```
Product::Product (
            int productId,
            QString name,
            double price,
            int quantity )
```

Construct a new Product object.

Parameters

productId	ID of the product.
name	Name of the product.
price	Price of the product. If negative, it will be set to 0.
quantity	Quantity of the product. If negative, it will be set to 0.

Definition at line 11 of file Product.cpp.

7.3.3 Member Function Documentation

7.3.3.1 changeQuantity()

```
status Product::changeQuantity (
            int quantity )
```

Change the quantity of the product.

Parameters

quantity	The new quantity of the product. Must be non-negative.

Returns

status SUCCESS if the operation is successful, ERROR otherwise.

Definition at line 76 of file Product.cpp.

7.3.3.2 getName()

```
QString Product::getName ( ) const
```

Get the name of the product.

Definition at line 91 of file Product.cpp.

7.3.3.3 getPrice()

```
double Product::getPrice ( ) const
```

Get the price of the product.

Definition at line 96 of file Product.cpp.

7.3.3.4 getQuantity()

```
int Product::getQuantity ( ) const
```

Get the quantity of the product.

Definition at line 101 of file Product.cpp.

7.3.3.5 sell()

```
status Product::sell (
            int quantityToSell )
```

Sell a quantity of the product.

Parameters

quantityToSell	The quantity of the product to sell.

Returns

status SUCCESS if the operation is successful, ERROR otherwise.

Definition at line 39 of file Product.cpp.

7.3.3.6 updatePrice()

```
status Product::updatePrice (
            double newPrice )
```

Update the price of the product.

Parameters

newPrice	The new price of the product. Must be positive.

Returns

status SUCCESS if the operation is successful, ERROR otherwise.

Definition at line 60 of file Product.cpp.

7.3.4 Member Data Documentation

7.3.4.1 productId

```
int Product::productId
```

ID of the product.

Definition at line 27 of file Product.h.

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

- src/Product/Product.h
- src/Product/Product.cpp

7.4 Report::ProductReport Struct Reference

Represents a report for a single product.

```
#include <Report.h>
```

Public Attributes

- QString name
  Name of the product.
- double price
  Price of the product.
- int quantity
  Quantity of the product.

7.4.1 Detailed Description

Represents a report for a single product.

This structure holds the details of a product for reporting purposes, including its name, price, and quantity.

Definition at line 31 of file Report.h.

7.4.2 Member Data Documentation

7.4.2.1 name

```
QString Report::ProductReport::name
```

Name of the product.

Definition at line 33 of file Report.h.

7.4.2.2 price

```
double Report::ProductReport::price
```

Price of the product.

Definition at line 34 of file Report.h.

7.4.2.3 quantity

```
int Report::ProductReport::quantity
```

Quantity of the product.

Definition at line 35 of file Report.h.

The documentation for this struct was generated from the following file:

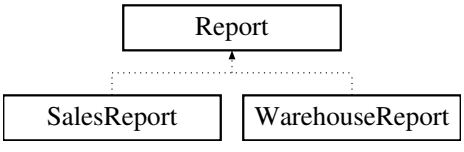
- src/Report/Report.h

7.5 Report Class Reference

Represents a report in the store simulation.

```
#include <Report.h>
```

Inheritance diagram for Report:



Classes

- struct ProductReport

  Represents a report for a single product.

Public Member Functions

- Report (double operationalCosts, double netProfit)

  Construct a new Report object.
- QString generateReport () const

  Generates a summary report as a QString.

Static Public Member Functions

- static void setOperationalCosts (double costs)

  Set the operational costs.
- static void setNetProfit (double profit)

  Set the net profit.
- static double getOperationalCosts ()

  Get the operational costs.
- static double getNetProfit ()

  Get the net profit.

7.5.1 Detailed Description

Represents a report in the store simulation.

This class is responsible for generating reports that summarize the operational costs and net profits.

Definition at line 17 of file Report.h.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 Report()

```
Report::Report (
            double operationalCosts,
            double netProfit )
```

Construct a new Report object.

Parameters

operationalCosts	Operational costs of the store.
netProfit	Net profit of the store.

Definition at line 17 of file Report.cpp.

7.5.3 Member Function Documentation

7.5.3.1 generateReport()

```
QString Report::generateReport ( ) const
```

Generates a summary report as a QString.

This method compiles the operational costs and net profits into a readable format.

Returns

QString A summary report of the store's financial status.

QString A summary report of the store's financial status.

Definition at line 28 of file Report.cpp.

7.5.3.2 getNetProfit()

```
double Report::getNetProfit ( )  [static]
```

Get the net profit.

Definition at line 52 of file Report.cpp.

7.5.3.3 getOperationalCosts()

```
double Report::getOperationalCosts ( )  [static]
```

Get the operational costs.

Definition at line 47 of file Report.cpp.

7.5.3.4 setNetProfit()

```
void Report::setNetProfit (
            double profit )  [static]
```

Set the net profit.

Definition at line 41 of file Report.cpp.

7.5.3.5 setOperationalCosts()

```
void Report::setOperationalCosts (
            double costs )  [static]
```

Set the operational costs.

Definition at line 36 of file Report.cpp.

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

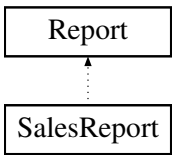
- src/Report/Report.h
- src/Report/Report.cpp

7.6 SalesReport Class Reference

The SalesReport class extends the Report class to provide a report specifically for sales.

`#include <SalesReport.h>`

Inheritance diagram for SalesReport:



Public Member Functions

- SalesReport (int salesId, QDateTime time, QList< ProductReport > productList, double operationalCosts, double netProfit)

  Construct a new SalesReport object.
- QString generateReport () const

  Generates a detailed sales report.

7.6.1 Detailed Description

The SalesReport class extends the Report class to provide a report specifically for sales.

This class inherits from Report and is responsible for generating a sales report, which includes details such as sales ID, time, and a list of products sold.

Definition at line 20 of file SalesReport.h.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 SalesReport()

```
SalesReport::SalesReport (
            int salesId,
            QDateTime time,
            QList< ProductReport > productList,
            double operationalCosts,
            double netProfit )
```

Construct a new SalesReport object.

Parameters

salesId	Unique identifier for the sales report.
time	The time when the report is generated.
productList	List of products included in the sales report.
operationalCosts	Operational costs of the store.
netProfit	Net profit of the store.

Definition at line 11 of file SalesReport.cpp.

7.6.3 Member Function Documentation

7.6.3.1 generateReport()

```
QString SalesReport::generateReport ( ) const
```

Generates a detailed sales report.

Returns

QString A formatted string representing the sales report.

Definition at line 21 of file SalesReport.cpp.

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

- src/SalesReport/SalesReport.h
- src/SalesReport/SalesReport.cpp

7.7 Simulation Class Reference

The Simulation class manages the overall department store simulation.

```
#include <Simulation.h>
```

Public Member Functions

- Simulation ()
  Construct a new Simulation object.
- status conductCycle ()
  Conducts a simulation cycle, processing events for the current cycle.
- status respondToEvent (Event &event)
  Responds to a specific event.
- void run ()
  Initiates the simulation by running the main loop.
- status processEvents ()
  Processes all scheduled events.
- QString generateReport ()
  Generates a report summarizing simulation results.

7.7.1 Detailed Description

The Simulation class manages the overall department store simulation.

This class orchestrates the simulation process, including event handling, warehouse management, product operations, and reporting.

Definition at line 28 of file Simulation.h.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 Simulation()

```
Simulation::Simulation ( )
```

Construct a new Simulation object.

Sets up the initial state of the simulation, including loading settings and preparing the environment.

Definition at line 11 of file Simulation.cpp.

7.7.3 Member Function Documentation

7.7.3.1 conductCycle()

```
status Simulation::conductCycle ( )
```

Conducts a simulation cycle, processing events for the current cycle.

Processes events and updates the simulation state for each cycle.

Returns

Status of the cycle (success or failure).

Definition at line 191 of file Simulation.cpp.

7.7.3.2 generateReport()

```
QString Simulation::generateReport ( )
```

Generates a report summarizing simulation results.

Compiles data from the simulation into a formatted report string.

Returns

QString A summary report of the simulation results.

Definition at line 332 of file Simulation.cpp.

7.7.3.3 processEvents()

```
status Simulation::processEvents ( )
```

Processes all scheduled events.

Invokes conductCycle to process events for the current cycle.

Returns

Status of proccessing events).

Definition at line 178 of file Simulation.cpp.

7.7.3.4 respondToEvent()

```
status Simulation::respondToEvent (
            Event & event )
```

Responds to a specific event.

Handles the given event based on its type and updates the simulation state accordingly.

Parameters

event	The event to handle.

Returns

Status of event handling (success or failure).

Definition at line 229 of file Simulation.cpp.

7.7.3.5 run()

```
void Simulation::run ( )
```

Initiates the simulation by running the main loop.

Starts the simulation loop, generating cycles, processing events, and generating reports.

Definition at line 110 of file Simulation.cpp.

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

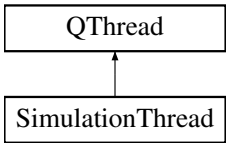
- src/Simulation/Simulation.h
- src/Simulation/Simulation.cpp

7.8 SimulationThread Class Reference

The SimulationThread class is responsible for running the simulation in a separate thread.

```
#include <gui.h>
```

Inheritance diagram for SimulationThread:



Signals

- void simulationFinished ()

  Signal emitted when the simulation has finished running.

Public Member Functions

- void run () override

  Executes the simulation in a separate thread.

7.8.1 Detailed Description

The SimulationThread class is responsible for running the simulation in a separate thread.

This class inherits from QThread and overrides the run method to execute the simulation. It emits a signal upon completion of the simulation to notify other components of its completion.

Definition at line 38 of file gui.h.

7.8.2 Member Function Documentation

7.8.2.1 run()

```
void SimulationThread::run ( )  [inline], [override]
```

Executes the simulation in a separate thread.

This function overrides the run method from QThread and is called when the thread starts. It initializes a Simulation object and runs the simulation. Upon completion, it emits the simulationFinished signal.

Definition at line 49 of file gui.h.

7.8.2.2 simulationFinished

```
void SimulationThread::simulationFinished ( )  [signal]
```

Signal emitted when the simulation has finished running.

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

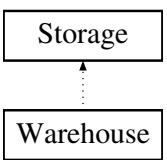
- src/gui/gui.h

7.9 Storage Class Reference

The Storage class represents a storage unit with a certain capacity.

```
#include <Storage.h>
```

Inheritance diagram for Storage:



Public Member Functions

- Storage (int capacity)
  Construct a new Storage object with a specified capacity.
- storageStatus checkCapacity (int totalCapacity) const
  Check the current capacity status of the storage.

Protected Attributes

- int capacity
  Storage fill level.

7.9.1 Detailed Description

The Storage class represents a storage unit with a certain capacity.

This class provides a method to check the current status of the storage based on its capacity.

Definition at line 19 of file Storage.h.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 Storage()

```
Storage::Storage (
            int capacity )
```

Construct a new Storage object with a specified capacity.

Parameters

capacity	The initial capacity of the storage.

Definition at line 11 of file Storage.cpp.

7.9.3 Member Function Documentation

7.9.3.1 checkCapacity()

```
storageStatus Storage::checkCapacity (
            int totalCapacity ) const
```

Check the current capacity status of the storage.

Parameters

totalCapacity	The maximum capacity of the storage.

Returns

storageStatus The status of the storage: EMPTY, AVAILABLE, or FULLY.

Definition at line 19 of file Storage.cpp.

7.9.4 Member Data Documentation

7.9.4.1 capacity

```
int Storage::capacity  [protected]
```

Storage fill level.

Definition at line 22 of file Storage.h.

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

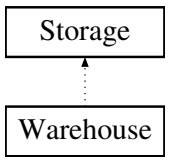
- src/Storage/Storage.h
- src/Storage/Storage.cpp

7.10 Warehouse Class Reference

The Warehouse class represents a warehouse with storage capacity.

```
#include <Warehouse.h>
```

Inheritance diagram for Warehouse:



Public Member Functions

- Warehouse (QString location, double warehouseCapacity)
  Construct a new Warehouse object.
- storageStatus checkStatus ()
  Check the current status of the warehouse and updates the capacity attribute inherited from the Storage class.
- status updateStatus (int newCapacity)
  Update the storage capacity of the warehouse.
- status addProduct (QString name, double price, int quantity, int productId)
  Add a new product to the warehouse.
- status updatePrice (double newPrice, int productId)
  Update the price of a product in the warehouse.
- status changeQuantity (int quantity, int productId)
  Change the quantity of a product in the warehouse.

- status sell (int quantityToSell, int productId)

  Sell a quantity of a product from the warehouse.
- QString getName (int productId)

  Get the name of a product by its ID.
- double getPrice (int productId)

  Get the price of a product by its ID.
- int getQuantity (int productId)

  Get the quantity of a product by its ID.
- QString getLocation () const

  Get the location of the warehouse.
- const QList< Product > getProductList () const

  Get the list of the products.
- double getCurrentCapacity () const

  Get the capacity of the warehouse.

Static Public Attributes

- static int warehouseId = 0

  Unique identifier for the warehouse.

7.10.1 Detailed Description

The Warehouse class represents a warehouse with storage capacity.

This class inherits from Storage and provides additional functionality specific to warehouse operations.

Definition at line 20 of file Warehouse.h.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 Warehouse()

```
Warehouse::Warehouse (
            QString location,
            double warehouseCapacity )
```

Construct a new Warehouse object.

Parameters

location	The location of the warehouse.
warehouseCapacity	The initial capacity of the warehouse.

Definition at line 14 of file Warehouse.cpp.

7.10.3 Member Function Documentation

7.10.3.1 addProduct()

```
status Warehouse::addProduct (
            QString name,
            double price,
            int quantity,
            int productId )
```

Add a new product to the warehouse.

Parameters

name	The name of the product.
price	The price of the product.
quantity	The quantity of the product.
productId	ID of the product.

Returns

status SUCCESS if the product is added, ERROR otherwise.

Definition at line 84 of file Warehouse.cpp.

7.10.3.2 changeQuantity()

```
status Warehouse::changeQuantity (
            int quantity,
            int productId )
```

Change the quantity of a product in the warehouse.

Parameters

quantity	The new quantity of the product.
productId	The ID of the product.

Returns

status SUCCESS if the quantity is updated, ERROR otherwise.

Definition at line 133 of file Warehouse.cpp.

7.10.3.3 checkStatus()

```
storageStatus Warehouse::checkStatus ( )
```

Check the current status of the warehouse and updates the capacity attribute inherited from the Storage class.

Check the current status of the warehouse and update the capacity attribute inherited from the Storage class.

Returns

storageStatus The status of the warehouse: EMPTY, AVAILABLE, or FULL.

Definition at line 33 of file Warehouse.cpp.

7.10.3.4 getCurrentCapacity()

```
double Warehouse::getCurrentCapacity ( ) const
```

Get the capacity of the warehouse.

Get the current capacity of the warehouse.

Definition at line 263 of file Warehouse.cpp.

7.10.3.5 getLocation()

```
QString Warehouse::getLocation ( ) const
```

Get the location of the warehouse.

Definition at line 247 of file Warehouse.cpp.

7.10.3.6 getName()

```
QString Warehouse::getName (
            int productId )
```

Get the name of a product by its ID.

Parameters

productId	The ID of the product.

Returns

QString The name of the product.

< Contains false if the product was not found in the productList.

Definition at line 184 of file Warehouse.cpp.

7.10.3.7 getPrice()

```
double Warehouse::getPrice (
            int productId )
```

Get the price of a product by its ID.

Parameters

productId	The ID of the product.

Returns

double The price of the product.

< Contains false if the product was not found in the productList.

Definition at line 205 of file Warehouse.cpp.

7.10.3.8 getProductList()

```
const QList< Product > Warehouse::getProductList ( ) const
```

Get the list of the products.

Get the list of the products stored in the warehouse.

Definition at line 255 of file Warehouse.cpp.

7.10.3.9 getQuantity()

```
int Warehouse::getQuantity (
            int productId )
```

Get the quantity of a product by its ID.

Parameters

productId	The ID of the product.

Returns

int The quantity of the product.

< Contains false if the product was not found in the productList.

Definition at line 226 of file Warehouse.cpp.

7.10.3.10 sell()

```
status Warehouse::sell (
            int quantityToSell,
            int productId )
```

Sell a quantity of a product from the warehouse.

Parameters

quantityToSell	The quantity of the product to sell.
productId	The ID of the product.

Returns

status SUCCESS if the product is sold, ERROR otherwise.

Definition at line 169 of file Warehouse.cpp.

7.10.3.11 updatePrice()

```
status Warehouse::updatePrice (
            double newPrice,
            int productId )
```

Update the price of a product in the warehouse.

Parameters

newPrice	The new price of the product.
productId	The ID of the product.

Returns

status SUCCESS if the price is updated, ERROR otherwise.

< Contains false if the product was not found in the productList.

Definition at line 112 of file Warehouse.cpp.

7.10.3.12 updateStatus()

```
status Warehouse::updateStatus (
            int newCapacity )
```

Update the storage capacity of the warehouse.

Parameters

newCapacity	The new capacity of the warehouse.

Returns

status SUCCESS if the operation is successful, ERROR otherwise.

Definition at line 60 of file Warehouse.cpp.

7.10.4 Member Data Documentation

7.10.4.1 warehouseId

```
int Warehouse::warehouseId = 0  [static]
```

Unique identifier for the warehouse.

Static variable initialization.

Definition at line 35 of file Warehouse.h.

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

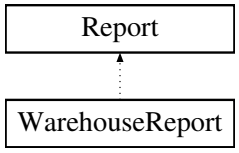
- src/Warehouse/Warehouse.h
- src/Warehouse/Warehouse.cpp

7.11 WarehouseReport Class Reference

The WarehouseReport class extends the Report class to provide a report specifically for warehouse inventory.

`#include <WarehouseReport.h>`

Inheritance diagram for WarehouseReport:



Public Member Functions

- WarehouseReport (int warehouseId, double capacity, QList< ProductReport > productList, double operationalCosts, double netProfit)

  Construct a new WarehouseReport object.
- QString generateReport () const

  Generates a detailed warehouse inventory report.

7.11.1 Detailed Description

The WarehouseReport class extends the Report class to provide a report specifically for warehouse inventory.

This class inherits from Report and is responsible for generating a warehouse report, which includes details such as warehouse ID, capacity, and a list of products stored.

Definition at line 19 of file WarehouseReport.h.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 WarehouseReport()

```
WarehouseReport::WarehouseReport (
            int warehouseId,
            double capacity,
            QList< ProductReport > productList,
            double operationalCosts,
            double netProfit )
```

Construct a new WarehouseReport object.

Parameters

warehouseId	Unique identifier for the warehouse.
capacity	Total storage capacity of the warehouse.
productList	List of products stored in the warehouse.
operationalCosts	Operational costs of the store.
netProfit	Net profit of the store.

Definition at line 11 of file WarehouseReport.cpp.

7.11.3 Member Function Documentation

7.11.3.1 generateReport()

```
QString WarehouseReport::generateReport ( ) const
```

Generates a detailed warehouse inventory report.

Returns

A formatted string representing the warehouse inventory report.

Definition at line 21 of file WarehouseReport.cpp.

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

- src/WarehouseReport/WarehouseReport.h
- src/WarehouseReport/WarehouseReport.cpp

Chapter 8

File Documentation

8.1 README.md File Reference

8.2 src/Event/Event.cpp File Reference

Source file of the Event class.

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

8.2.1 Detailed Description

Source file of the Event class.

Definition in file Event.cpp.

8.3 Event.cpp

Go to the documentation of this file.

```
00001
00006 #include "Event.h"
00007 #include <QRandomGenerator>
00008
00012 Event::Event(QString eventType, QDateTime time)
00013 {
00014     this -> eventType = eventType;
00015     this -> time = time;
00016 }
00017
00025 Event Event::generateEvent(QString eventType, int seed)
00026 {
00027     QDateTime time = QDateTime::currentDateTime().addMSecs(QRandomGenerator::global() ->
      bounded(seed));
00028     return Event(eventType, time);
00029 }
00030
00031 // Getters implementation
00032
00038 QString Event::getEventType() const
00039 {
00040     return eventType;
00041 }
00042
00048 QDateTime Event::getTime() const
00049 {
00050     return time;
00051 }
```

8.4 src/Event/Event.h File Reference

Header file of the Event class.

```
#include <QDateTime>
```

Classes

- class Event

  The Event class represents an event in the warehouse simulation.

8.4.1 Detailed Description

Header file of the Event class.

Definition in file Event.h.

8.5 Event.h

Go to the documentation of this file.

```
00001
00006 #ifndef EVENT_H
00007 #define EVENT_H
00008
00009 #include <QDateTime>
00010
00017 class Event
00018 {
00019 private:
00020     QString eventType;
00021     QDateTime time;
00022
00023 public:
00031     Event(QString eventType, QDateTime time);
00032
00041     static Event generateEvent(QString eventType, int seed);
00042
00043     // Getters
00044     QString getEventType() const;
00045     QDateTime getTime() const;
00046 };
00047
00048 #endif // EVENT_H
```

8.6 src/gui/gui.cpp File Reference

Source file for the GUI class.

```
#include "gui.h"
#include "./ui_gui.h"
```

8.6.1 Detailed Description

Source file for the GUI class.

Definition in file gui.cpp.

8.7 gui.cpp

Go to the documentation of this file.

```
00001
00006 #include "gui.h"
00007 #include "./ui_gui.h"
00008
00014 GUI::GUI(GUIElement *parent)
00015     : QMainWindow(parent)
00016     , ui(new Ui::GUI)
00017 {
00018     ui->setupUi(this);
00019     render();
00020 }
00021
00025 GUI::~GUI()
00026 {
00027     delete ui;
00028 }
00029
00033 void GUI::render()
00034 {
00035     ui -> stackedWidget -> setCurrentIndex(0);
00036
00037     //Warehouse page
00038     Warehouse warehouse;
00039     warehouse.id = 1;
00040     warehouses.append(warehouse);
00041     loadCurrentWarehouseData();
00042
00043     ui->productTable->setEditTriggers(QAbstractItemView::NoEditTriggers);
00044     ui->productTable->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);
00045     ui->productTable->setStyleSheet("QHeaderView::section { font-size: 15pt; }");
00046
00047     //Settings page
00048     ui -> configFileEdit -> setDisabled(true);
00049     ui -> seedEdit -> setText(QString::number(seed));
00050     ui -> cyclesEdit -> setText(QString::number(cycles));
00051     ui -> configFileEdit -> setText(filename);
00052
00053     //Simulation page
00054     ui->statisticList->setEditTriggers(QAbstractItemView::NoEditTriggers);
00055     ui->statisticList->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);
00056     ui->statisticList->setStyleSheet("QHeaderView::section { font-size: 15pt; }");
00057
00058     ui->raportList->setEditTriggers(QAbstractItemView::NoEditTriggers);
00059     ui->raportList->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);
00060     ui->raportList->setStyleSheet("QHeaderView::section { font-size: 15pt; }");
00061 }
00062
00069 void GUI::on_addbutton_clicked()
00070 {
00071     if(warehouses[0].name == "")
00072     {
00073         warehouses[0].name = QInputDialog::getText(this, "Add warehouse", "Please enter warehouse
      location below: ");
00074         warehouses[0].capacity = QInputDialog::getInt(this, "Add warehouse", "Please enter warehouse
      capacity below: ");
00075     }
00076     ui->stackedWidget->setCurrentIndex(1);
00077 }
00078
00082 void GUI::on_back_to_menu_clicked()
00083 {
00084     ui->stackedWidget->setCurrentIndex(0);
00085 }
00086
00090 void GUI::on_back_to_menu_2_clicked()
00091 {
00092     ui->stackedWidget->setCurrentIndex(0);
00093 }
00094
```

```
00098 void GUI::on_back_to_menu_3_clicked()
00099 {
00100     ui->stackedWidget->setCurrentIndex(0);
00101 }
00102
00106 void GUI::on_settings_button_clicked()
00107 {
00108     ui->stackedWidget->setCurrentIndex(2);
00109 }
00110
00114 void GUI::on_start_simulation_button_clicked()
00115 {
00116     ui->stackedWidget->setCurrentIndex(3);
00117 }
00118
00122 void GUI::on_aboutButton_clicked()
00123 {
00124     QApplication::aboutQt();
00125 }
00126
00130 void GUI::on_configFileButton_clicked()
00131 {
00132     filename = QFileDialog::getOpenFileName(this, "Choose setting file", "settings.csv", "*.csv");
00133     if(filename == "")
00134     {
00135         QMessageBox::information(this, "Settings ", "No file selected.");
00136         ui -> configFileEdit -> setText(filename);
00137     }
00138     else
00139     {
00140         ui -> configFileEdit -> setText(filename);
00141
00142         QFile file(filename);
00143         if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
00144             QMessageBox::critical(this, "Load Data", "Cannot open file.");
00145             return;
00146         }
00147
00148         QTextStream in(&file);
00149
00150         warehouses.clear();
00151
00152         Warehouse *currentWarehouse = nullptr;
00153
00154         while (!in.atEnd())
00155         {
00156             QString line = in.readLine();
00157             QStringList fields = line.split(",");
00158             if (fields[0] == "Warehouse")
00159             {
00160                 Warehouse warehouse;
00161                 warehouse.id = warehouses.size() + 1;
00162                 warehouse.name = fields[1];
00163                 warehouse.capacity = fields[2].toInt();
00164                 warehouses.append(warehouse);
00165                 currentWarehouse = &warehouses.last();
00166             }
00167             else if (fields[0] == "Product" && currentWarehouse)
00168             {
00169                 if (fields.size() < 4)
00170                 {
00171                     QMessageBox::critical(this, "Load Data", "Incorrect product line format.");
00172                     continue;
00173                 }
00174                 Product product;
00175                 product.name = fields[1];
00176                 product.price = fields[2].toDouble();
00177                 product.quantity = fields[3].toInt();
00178                 currentWarehouse->products.append(product);
00179             }
00180             else if (fields[0] == "Seed")
00181             {
00182                 seed = fields[5].toInt();
00183                 ui -> seedEdit -> setText(QString::number(seed));
00184             }
00185             else if (fields[0] == "Cycles")
00186             {
00187                 cycles = fields[4].toInt();
00188                 ui -> cyclesEdit -> setText(QString::number(cycles));
00189             }
00190         }
00191         file.close();
00192         loadCurrentWarehouseData();
00193     }
00194 }
00195
00199 void GUI::on_back_to_menu_7_clicked()
```

```
00200 {
00201     seed = ui -> seedEdit -> text().toInt();
00202     cycles = ui -> cyclesEdit -> text().toInt();
00203     ui->stackedWidget->setCurrentIndex(0);
00204 }
00205
00209 void GUI::on_start_button_clicked()
00210 {
00211     ui -> statisticList -> setRowCount(0);
00212
00213     if(warehouses.isEmpty() || seed == 0 || cycles == 0 || warehouses[0].products.isEmpty())
00214     {
00215         QMessageBox::warning(this, "Start Simulation", "Cannot start simulation without initial
      conditions.");
00216         return;
00217     }
00218
00219     QFile file("settings.csv");
00220     if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {
00221         QMessageBox::critical(this, "Save Data", "Cannot open file.");
00222         return;
00223     }
00224
00225     QTextStream out(&file);
00226     out « "Type,Location,Capacity,Name,Price,Quantity,Cycles,Seed\n";
00227
00228     for (const Warehouse &warehouse : warehouses)
00229     {
00230         out « "Warehouse," « warehouse.name « "," « warehouse.capacity « "\n";
00231
00232         for (const Product &product : warehouse.products)
00233         {
00234             int row = ui -> statisticList -> rowCount();
00235             ui -> statisticList -> insertRow(row);
00236             ui -> statisticList -> setItem(row, 0, new
      QTableWidgetItem(QString::number(warehouse.id-1)));
00237             ui -> statisticList -> setItem(row, 1, new QTableWidgetItem(QString::number(row+1)));
00238             ui -> statisticList -> setItem(row, 2, new QTableWidgetItem(product.name));
00239             ui -> statisticList -> setItem(row, 3, new
      QTableWidgetItem(QString::number(product.price)));
00240             ui -> statisticList -> setItem(row, 4, new
      QTableWidgetItem(QString::number(product.quantity)));
00241             out « "Product," « product.name « "," « product.price « "," « product.quantity « "\n";
00242         }
00243         out « "Cycles,," « cycles « "\n";
00244         out « "Seed,,," « seed « "\n";
00245     }
00246     file.close();
00247
00248     SimulationThread *simulationThread = new SimulationThread();
00249     connect(simulationThread, &SimulationThread::finished, simulationThread, &QObject::deleteLater);
00250     connect(simulationThread, &SimulationThread::simulationFinished, this,
      &GUI::onSimulationFinished);
00251     simulationThread->start();
00252
00253     setupUpdateTimer();
00254     updateTablesFromCSV();
00255 }
00256
00260 void GUI::on_addProductButton_clicked()
00261 {
00262     QString name = ui->productNameEntry->text();
00263     QString price = ui->productPriceEntry->text();
00264     QString quantity = ui->productQuantityEntry->text();
00265
00266     bool priceValidation, quantityValidation;
00267     double priceValue = price.toDouble(&priceValidation);
00268     int quantityValue = quantity.toInt(&quantityValidation);
00269
00270     Warehouse &currentWarehouse = warehouses[currentWarehouseIndex];
00271     int currentCapacity = 0;
00272     for (const Product &product : currentWarehouse.products)
00273     {
00274         currentCapacity += product.quantity;
00275     }
00276
00277     if(!priceValidation || !quantityValidation || priceValue < 0 || quantityValue < 0 ||
      currentCapacity + quantityValue > currentWarehouse.capacity)
00278     {
00279         QMessageBox::warning(this, "Add Product", "Wrong product parameters or exceeding warehouse
      capacity.");
00280         return;
00281     }
00282
00283     Product newProduct{name, priceValue, quantityValue};
00284     currentWarehouse.products.append(newProduct);
00285     loadCurrentWarehouseData();
```

```
00286 }
00287
00288 void GUI::on_removeButton_clicked()
00289 {
00290     QPushButton *button = qobject_cast<QPushButton *>(sender());
00291     if(button)
00292     {
00293         int row = button->property("row").toInt();
00294         ui->productTable->removeRow(row);
00295
00296         Warehouse &currentWarehouse = warehouses[currentWarehouseIndex];
00297         if(row < currentWarehouse.products.size())
00298         {
00299             currentWarehouse.products.removeAt(row);
00300         }
00301
00302         for(int currentRow = row; currentRow < ui->productTable->rowCount(); ++currentRow)
00303         {
00304             QPushButton *btnRemove = qobject_cast<QPushButton
      *>(ui->productTable->cellWidget(currentRow, 3));
00305             btnRemove->setProperty("row", currentRow);
00306         }
00307
00308         ui->productTable->update();
00309     }
00310 }
00311
00315 void GUI::loadCurrentWarehouseData()
00316 {
00317     ui->productTable->setRowCount(0);
00318     Warehouse &currentWarehouse = warehouses[currentWarehouseIndex];
00319
00320     ui->warehouseLabel->setText("Warehouse " + QString::number(currentWarehouse.id));
00321
00322     for(const Product &product : currentWarehouse.products)
00323     {
00324         int currentRow = ui->productTable->rowCount();
00325         ui->productTable->insertRow(currentRow);
00326
00327         ui->productTable->setItem(currentRow, 0, new QTableWidgetItem(product.name));
00328         ui->productTable->setItem(currentRow, 1, new
      QTableWidgetItem(QString::number(product.price)));
00329         ui->productTable->setItem(currentRow, 2, new
      QTableWidgetItem(QString::number(product.quantity)));
00330
00331         QPushButton *btnRemove = new QPushButton("Remove");
00332         btnRemove->setProperty("row", currentRow);
00333         connect(btnRemove, &QPushButton::clicked, this, &GUI::on_removeButton_clicked);
00334         ui->productTable->setCellWidget(currentRow, 3, btnRemove);
00335     }
00336 }
00337
00341 void GUI::on_previousWarehouse_clicked()
00342 {
00343     if (currentWarehouseIndex > 0)
00344     {
00345         currentWarehouseIndex--;
00346     }
00347     else
00348     {
00349         currentWarehouseIndex = warehouses.size() - 1;
00350     }
00351     loadCurrentWarehouseData();
00352 }
00353
00357 void GUI::on_nextWarehouse_clicked()
00358 {
00359     if (currentWarehouseIndex == warehouses.size() - 1)
00360     {
00361         Warehouse newWarehouse;
00362         newWarehouse.id = warehouses.size() + 1;
00363         newWarehouse.name = QInputDialog::getText(this, "Add warehouse", "Please enter warehouse
      location below: ");
00364         newWarehouse.capacity = QInputDialog::getInt(this, "Add warehouse", "Please enter warehouse
      capacity below: ");
00365         warehouses.append(newWarehouse);
00366         currentWarehouseIndex = warehouses.size() - 1;
00367     }
00368     else
00369     {
00370         currentWarehouseIndex = (currentWarehouseIndex + 1) % warehouses.size();
00371     }
00372     loadCurrentWarehouseData();
00373 }
00374
00378 void GUI::on_removeWarehouseButton_clicked()
00379 {
```

```
00380     if (warehouses.size() > 1 && currentWarehouseIndex < warehouses.size())
00381     {
00382         warehouses.removeAt(currentWarehouseIndex);
00383         if (currentWarehouseIndex == warehouses.size())
00384         {
00385             currentWarehouseIndex--;
00386         }
00387         loadCurrentWarehouseData();
00388     }
00389     else
00390     {
00391         QMessageBox::warning(this, "Remove Warehouse", "Cannot remove the only warehouse.");
00392     }
00393 }
00394
00398 void GUI::setupUpdateTimer()
00399 {
00400     timer = new QTimer(this);
00401     connect(timer, &QTimer::timeout, this, &GUI::updateTablesFromCSV);
00402     timer->start(1000);
00403 }
00404
00408 void GUI::updateTablesFromCSV()
00409 {
00410     ui -> raportList -> setRowCount(0);
00411     int cycles = 0;
00412     QFile report("SimulationReport.csv");
00413
00414     if(!report.open(QIODevice::ReadOnly | QIODevice::Text))
00415     {
00416         QMessageBox::critical(this, "Read Report", "Cannot open report file to read.");
00417         return;
00418     }
00419
00420     int currentWarehouse;
00421     int statisticRow = 0;
00422
00423     QTextStream in(&report);
00424
00425     while(!in.atEnd())
00426     {
00427         QString line = in.readLine();
00428         if(line.isEmpty())
00429         {
00430             statisticRow = 0;
00431             cycles++;
00432         }
00433         else if(line.contains("Warehouse ID,Capacity"))
00434         {
00435             line = in.readLine();
00436
00437             QStringList fields = line.split(',');
00438             currentWarehouse = fields[0].toInt();
00439             int capacity = fields[1].toInt();
00440             QString location = warehouses[currentWarehouse].name;
00441             int sold = 0;
00442
00443             line = in.readLine();
00444             if(!line.contains("Product Name,Price,Quantity"))
00445             {
00446                 break;
00447             }
00448             line = in.readLine();
00449             while(!line.contains("Sales ID,Time"))
00450             {
00451                 QString productName;
00452                 double price;
00453                 int quantity;
00454
00455                 fields = line.split(",");
00456
00457                 productName = fields[0];
00458                 price = fields[1].toDouble();
00459                 quantity = fields[2].toInt();
00460
00461                 ui -> statisticList -> item(statisticRow, 2) -> setText(productName);
00462                 ui -> statisticList -> item(statisticRow, 3) -> setText(QString::number(price));
00463                 ui -> statisticList -> item(statisticRow, 4) -> setText(QString::number(quantity));
00464                 ui -> statisticList -> update();
00465                 statisticRow++;
00466                 line = in.readLine();
00467             }
00468             line = in.readLine();
00469             line = in.readLine();
00470             if(!line.contains("Product Name,Price,Quantity"))
00471             {
00472                 break;
```

```
00473             }
00474             line = in.readLine();
00475             while(!line.contains("Operational Costs,Net Profit"))
00476             {
00477                 fields = line.split(",");
00478                 sold += fields[2].toInt();
00479                 line = in.readLine();
00480             }
00481 
00482             line = in.readLine();
00483             fields = line.split(",");
00484 
00485             double costs = fields[0].toDouble();
00486             double netProfit = fields[1].toDouble();
00487 
00488             int row = ui -> raportList -> rowCount();
00489             ui -> raportList -> insertRow(row);
00490             ui -> raportList -> setItem(row, 0, new QTableWidgetItem(QString::number(cycles)));
00491             ui -> raportList -> setItem(row, 1, new
      QTableWidgetItem(QString::number(currentWarehouse)));
00492             ui -> raportList -> setItem(row, 2, new QTableWidgetItem(location));
00493             ui -> raportList -> setItem(row, 3, new QTableWidgetItem(QString::number(sold)));
00494             ui -> raportList -> setItem(row, 4, new QTableWidgetItem(QString::number(capacity)));
00495             ui -> raportList -> setItem(row, 5, new QTableWidgetItem(QString::number(costs)));
00496             ui -> raportList -> setItem(row, 6, new QTableWidgetItem(QString::number(netProfit)));
00497             ui -> raportList -> update();
00498         }
00499     }
00500     report.close();
00501 }
00502 
00506 void GUI::onSimulationFinished()
00507 {
00508     timer -> stop();
00509     updateTablesFromCSV();
00510 
00511     QMessageBox::StandardButton reply;
00512     reply = QMessageBox::question(this, "Save Report", "Do you want to save the report in a location
      other than the default one?", QMessageBox::Yes|QMessageBox::No);
00513 
00514     QString filePath;
00515     if (reply == QMessageBox::Yes)
00516     {
00517         filePath = QFileDialog::getSaveFileName(this, "Save Report", "SimulationReport.csv", "*.csv");
00518         if (!filePath.isEmpty())
00519         {
00520             QFile::copy("SimulationReport.csv", filePath);
00521         }
00522     }
00523 
00524     reply = QMessageBox::question(this, "Save data", "Do you want to save data from the table
      'Raport'?", QMessageBox::Yes|QMessageBox::No);
00525 
00526     if (reply == QMessageBox::Yes)
00527     {
00528         filePath = QFileDialog::getSaveFileName(this, "Save Report", "GUIReport.csv", "*.csv");
00529 
00530         QFile file(filePath);
00531         if(file.open(QIODevice::WriteOnly | QIODevice::Text))
00532         {
00533             QTextStream stream(&file);
00534             stream << "Cycle,Warehouse ID,Location,Sold Products,Capacity,Operational Costs,Net
      Profit\n";
00535             for(int row = 0; row < ui->raportList->rowCount(); ++row)
00536             {
00537                 QStringList rowData;
00538                 for(int column = 0; column < ui->raportList->columnCount(); ++column)
00539                 {
00540                     QTableWidgetItem *item = ui->raportList->item(row, column);
00541                     if(item)
00542                     {
00543                         rowData << item -> text();
00544                     }
00545                     else
00546                     {
00547                         rowData << "";
00548                     }
00549                 }
00550                 stream << rowData.join(",") << "\n";
00551             }
00552             file.close();
00553         }
00554         else
00555         {
00556             QMessageBox::critical(this, "Export to CSV", "Cannot open file for writing.");
00557         }
00558     }
```

```
00559
00560     reply = QMessageBox::question(this, "Save Settings", "Do you want to save yours settings in a
      location other than the default one?", QMessageBox::Yes|QMessageBox::No);
00561
00562     if (reply == QMessageBox::Yes)
00563     {
00564         filePath = QFileDialog::getSaveFileName(this, "Save Settings", "settings.csv", "*.csv");
00565         if (!filePath.isEmpty())
00566         {
00567             QFile::copy("settings.csv", filePath);
00568         }
00569     }
00570 }
```

8.8 src/gui/gui.h File Reference

Header file for the GUI class.

```
#include <QMainWindow>
#include "Simulation/Simulation.h"
#include <QMessageBox>
#include <QFileDialog>
#include <QFile>
#include <QTextStream>
#include <QInputDialog>
#include <QTimer>
#include <QThread>
```

Classes

- class SimulationThread

  The SimulationThread class is responsible for running the simulation in a separate thread.
- class GUI

  The GUI class inherits from QMainWindow and represents the main window of the application.

Namespaces

- namespace Ui

Typedefs

- using GUIElement = QWidget

  Alias for QWidget representing GUI elements.

8.8.1 Detailed Description

Header file for the GUI class.

Declares the GUI class and its members, which manage the user interface for the application.

Definition in file gui.h.

8.8.2 Typedef Documentation

8.8.2.1 GUIElement

```
GUIElement = QWidget
```

Alias for QWidget representing GUI elements.

GUIElement is an alias for QWidget and represents the basic unit of user interface elements in Qt. It can be used to refer to any widget that is part of the GUI, such as buttons, labels, text fields, etc.

Definition at line 29 of file gui.h.

8.9 gui.h

Go to the documentation of this file.

```
00001
00008 #ifndef GUI_H
00009 #define GUI_H
00010
00011 #include <QMainWindow>
00012 #include "Simulation/Simulation.h"
00013 #include <QMessageBox>
00014 #include <QFileDialog>
00015 #include <QFile>
00016 #include <QTextStream>
00017 #include <QInputDialog>
00018 #include <QTimer>
00019 #include <QThread>
00020
00029 using GUIElement = QWidget;
00030
00038 class SimulationThread : public QThread
00039 {
00040     Q_OBJECT
00041 public:
00049     void run() override
00050     {
00051         Simulation simulation;
00052         simulation.run();
00053         emit simulationFinished();
00054     }
00055 signals:
00059     void simulationFinished();
00060 };
00061
00062 namespace Ui {
00063 class GUI;
00064 }
00065
00073 class GUI : public QMainWindow
00074 {
00075     Q_OBJECT
00076
00077 public:
00083     explicit GUI(GUIElement *parent = nullptr);
00084
00088     ~GUI();
00089
00097     void render();
00098
00099 private slots:
00106     void on_addbutton_clicked();
00107
00113     void on_back_to_menu_clicked();
00114
00121     void on_back_to_menu_2_clicked();
00122
00129     void on_back_to_menu_3_clicked();
00130
00136     void on_settings_button_clicked();
00137
00144     void on_start_simulation_button_clicked();
00145
```

```
00151     void on_aboutButton_clicked();
00152
00159     void on_configFileButton_clicked();
00160
00166     void on_back_to_menu_7_clicked();
00167
00174     void on_start_button_clicked();
00175
00182     void on_addProductButton_clicked();
00183
00189     void on_removeButton_clicked();
00190
00196     void on_previousWarehouse_clicked();
00197
00204     void on_nextWarehouse_clicked();
00205
00211     void on_removeWarehouseButton_clicked();
00212
00213 private:
00214     Ui::GUI *ui;
00215
00223     struct Product
00224     {
00225         QString name;
00226         double price;
00227         int quantity;
00228     };
00229
00237     struct Warehouse
00238     {
00239         int id;
00240         int capacity;
00241         QString name;
00242         QList<Product> products;
00243     };
00244
00245     QList<Warehouse> warehouses;
00246     int currentWarehouseIndex = 0;
00247
00248     QString filename;
00249     int seed = 100;
00250     int cycles = 10;
00251
00252     QTimer *timer;
00253
00260     void loadCurrentWarehouseData();
00261
00268     void setupUpdateTimer();
00269
00276     void updateTablesFromCSV();
00277
00284     void onSimulationFinished();
00285 };
00286
00287 #endif // GUI_H
```

8.10 src/main.cpp File Reference

Main entry point for the Warehouse Simulator application.

```
#include "gui/gui.h"
#include "Simulation/Simulation.h"
#include <iostream>
#include <QFile>
#include <QTextStream>
#include <QApplication>
```

Functions

- void createConfigFile ()

  Creates a configuration file for the simulation.
- int main (int argc, char *argv[])

  Main function of the application.

8.10.1 Detailed Description

Main entry point for the Warehouse Simulator application.

Definition in file main.cpp.

8.10.2 Function Documentation

8.10.2.1 createConfigFile()

```
void createConfigFile ( )
```

Creates a configuration file for the simulation.

This function prompts the user to configure the simulation settings and writes them to a CSV file named "settings.↩ csv".

Definition at line 19 of file main.cpp.

8.10.2.2 main()

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

Main function of the application.

This function initializes the application, processes command-line arguments, and starts the GUI or simulation based on the provided options.

Parameters

argc	Number of command-line arguments.
argv	Array of command-line arguments.

Returns

Exit code of the application.

Definition at line 155 of file main.cpp.

8.11 main.cpp

Go to the documentation of this file.

```
00001
00006 #include "gui/gui.h"
00007 #include "Simulation/Simulation.h"
00008 #include <iostream>
00009 #include <QFile>
```

```
00010 #include <QTextStream>
00011 #include <QApplication>
00012
00019 void createConfigFile()
00020 {
00021     QVector<QString> configLines;
00022     QString tempLine;
00023
00024     // Open a CSV file for writing.
00025     QFile file("settings.csv");
00026
00027     if(file.open(QIODevice::WriteOnly | QIODevice::Text))
00028     {
00029         QTextStream out(&file);
00030
00031         // Write the CSV header.
00032         out << "Type,Location,Capacity,Name,Price,Quantity,Cycles,Seed\n";
00033
00034         unsigned short option = 0;
00035         unsigned int seed = 0; // Default seed value
00036
00037         // Interactive menu to configure simulation settings.
00038         while(option != 9)
00039         {
00040             // Display options to the user.
00041             std::cout << "\n\n*************************" << std::endl
00042                       << "* 1. Add warehouse        *" << std::endl
00043                       << "* 2. Add product          *" << std::endl
00044                       << "* 3. Set number of cycles *" << std::endl
00045                       << "* 4. Set event seed       *" << std::endl
00046                       << "* 5. Undo last change     *" << std::endl
00047                       << "* 9. Exit configuration   *" << std::endl
00048                       << "*************************" << std::endl;
00049
00050             std::cout << "\n\nEnter option: "; std::cin >> option;
00051
00052             // Handle user input based on selected option.
00053             switch(option)
00054             {
00055             case 1:
00056             {
00057                 std::string location;
00058                 double capacity;
00059                 std::cout << "\n\nEnter warehouse location: "; std::cin >> location;
00060                 std::cout << "Enter capacity of warehouse: "; std::cin >> capacity;
00061
00062                 // Write warehouse details to the config file.
00063                 tempLine = "Warehouse," + QString::fromStdString(location) + "," +
    QString::number(capacity);
00064                 configLines.push_back(tempLine);
00065                 break;
00066             }
00067             case 2:
00068             {
00069                 std::string name;
00070                 double price;
00071                 int quantity;
00072                 std::cout << "\n\nEnter product name: "; std::cin >> name;
00073                 std::cout << "Enter product price: "; std::cin >> price;
00074                 std::cout << "Enter quantity: "; std::cin >> quantity;
00075
00076                 // Write product details to the config file.
00077                 tempLine = "Product," + QString::fromStdString(name) + "," + QString::number(price) +
    "," + QString::number(quantity);
00078                 configLines.push_back(tempLine);
00079                 break;
00080             }
00081             case 3:
00082             {
00083                 int cycles;
00084                 std::cout << "\n\nEnter number of cycles: "; std::cin >> cycles;
00085
00086                 // Write the number of cycles to the config file.
00087                 tempLine = "Cycles,," + QString::number(cycles);
00088                 configLines.push_back(tempLine);
00089                 break;
00090             }
00091             case 4:
00092             {
00093                 std::cout << "\n\nEnter seed for event generation: "; std::cin >> seed;
00094
00095                 // Append the seed to the configuration.
00096                 tempLine = "Seed,,," + QString::number(seed);
00097                 configLines.push_back(tempLine);
00098                 break;
00099             }
00100             case 5:
```

```
00101             {
00102                 // Undo the last change.
00103                 if(!configLines.isEmpty())
00104                 {
00105                     configLines.pop_back();
00106                     std::cout « "\nLast change undone.\n";
00107                 }
00108                 else
00109                 {
00110                     std::cout « "\nNo changes to undo.\n";
00111                 }
00112                 break;
00113             }
00114             case 9:
00115             {
00116                 // Exit the configuration menu.
00117                 std::cout « "\nExiting configuration.\n";
00118                 break;
00119             }
00120             default:
00121             {
00122                 // Handle invalid input.
00123                 std::cout « "\n\nInvalid option. Please try again.";
00124                 break;
00125             }
00126             }
00127         }
00128 
00129         // Write all lines to the CSV file.
00130         for(const QString &line : configLines)
00131         {
00132             out « line « "\n";
00133         }
00134 
00135         // Close the file after writing.
00136         file.close();
00137     }
00138     else
00139     {
00140         // Error handling if the file cannot be opened.
00141         std::cerr « "Error: Can't open file to write.";
00142     }
00143 }
00144 
00155 int main(int argc, char *argv[])
00156 {
00157     bool _gui = true;
00158     bool _config = true;
00159 
00160     // Process command-line arguments.
00161     for(int arg = 1; arg < argc; ++arg)
00162     {
00163         if(strcmp(argv[arg], "--nogui") == 0)
00164         {
00165             _gui = false;
00166         }
00167         else if(strcmp(argv[arg], "--noconfig") == 0)
00168         {
00169             _config = false;
00170         }
00171         else if(strcmp(argv[arg], "--file") == 0)
00172         {
00173             // Check if there is a filename argument following the --file flag.
00174             if(arg + 1 < argc)
00175             {
00176                 QString inputFileName = argv[arg + 1];
00177 
00178                 // Check if the file exists.
00179                 if(QFile::exists(inputFileName))
00180                 {
00181                     // Rename the existing configuration file if it exists.
00182                     if(QFile::exists("settings.csv"))
00183                     {
00184                         QFile::rename("settings.csv", "settings_old.csv");
00185                     }
00186 
00187                     // Copy the new configuration file.
00188                     QFile::remove("settings.csv");
00189                     QFile::copy(inputFileName, "settings.csv");
00190 
00191                     // Skip the filename argument so it's not processed as another flag.
00192                     ++arg;
00193 
00194                     _config = false;
00195                 }
00196                 else
00197                 {
```

```
00198                     std::cerr « "The specified file does not exist: " « inputFileName.toStdString();
00199                     return -1;
00200                 }
00201             }
00202             else
00203             {
00204                 std::cerr « "No filename was specified after the --file flag.";
00205                 return -1;
00206             }
00207         }
00208     }
00209
00210     QApplication a(argc, argv);
00211     GUI w;
00212
00213     // Start the GUI if enabled.
00214     if(_gui)
00215     {
00216         w.show();
00217     }
00218     else
00219     {
00220         // Create a config file and run the simulation if enabled.
00221         if(_config)
00222         {
00223             createConfigFile();
00224         }
00225
00226         Simulation simulation = Simulation();
00227         simulation.run();
00228         exit(0);
00229     }
00230
00231     // Execute the application.
00232     return a.exec();
00233 }
```

8.12 test/main.cpp File Reference

```
#include <gtest/gtest.h>
```

Functions

- int main (int argc, char ∗argv[])

8.12.1 Function Documentation

8.12.1.1 main()

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

Definition at line 3 of file main.cpp.

8.13 main.cpp

Go to the documentation of this file.

```
00001 #include <gtest/gtest.h>
00002
00003 int main(int argc, char *argv[]) {
00004     ::testing::InitGoogleTest(&argc, argv);
00005     return RUN_ALL_TESTS();
00006 }
```

8.14 src/Product/Product.cpp File Reference

Source file for the Product class.

```
#include "Product.h"
```

8.14.1 Detailed Description

Source file for the Product class.

Definition in file Product.cpp.

8.15 Product.cpp

Go to the documentation of this file.

```
00001
00006 #include "Product.h"
00007
00011 Product::Product(int productId, QString name, double price, int quantity)
00012 {
00013     this -> productId = productId;
00014
00015     this -> name = name;
00016
00017     if(price < 0)
00018     {
00019         this -> price = 0;
00020     }
00021     else
00022     {
00023         this -> price = price;
00024     }
00025
00026     if(quantity < 0)
00027     {
00028         this -> quantity = 0;
00029     }
00030     else
00031     {
00032         this -> quantity = quantity;
00033     }
00034 }
00035
00039 status Product::sell(int quantityToSell)
00040 {
00041     if(quantityToSell <= 0)
00042     {
00043         return ERROR; // Ensure quantityToSell is positive
00044     }
00045
00046     if(quantityToSell <= quantity)
00047     {
00048         quantity -= quantityToSell;
00049         return SUCCESS;
00050     }
00051     else
00052     {
00053         return ERROR;
00054     }
00055 }
00056
00060 status Product::updatePrice(double newPrice)
00061 {
00062     if(newPrice > 0)
00063     {
00064         price = newPrice;
00065         return SUCCESS;
00066     }
00067     else
00068     {
00069         return ERROR;
```

```
00070     }
00071 }
00072
00076 status Product::changeQuantity(int quantity)
00077 {
00078     if(quantity >= 0)
00079     {
00080         this -> quantity = quantity;
00081         return SUCCESS;
00082     }
00083     else
00084     {
00085         return ERROR;
00086     }
00087 }
00088
00089 // Getters implementation
00090
00091 QString Product::getName() const
00092 {
00093     return name;
00094 }
00095
00096 double Product::getPrice() const
00097 {
00098     return price;
00099 }
00100
00101 int Product::getQuantity() const
00102 {
00103     return quantity;
00104 }
00105
```

8.16 src/Product/Product.h File Reference

Header file for the Product class.

```
#include <QString>
```

Classes

- class Product

  The Product class represents a product with a name, price, and quantity.

Enumerations

- enum status { SUCCESS , ERROR }

8.16.1 Detailed Description

Header file for the Product class.

Definition in file Product.h.

8.16.2 Enumeration Type Documentation

8.16.2.1 status

```
enum status
```

Enum representing the status of operations on Product.

Enumerator

SUCCESS	
ERROR	

Definition at line 12 of file Product.h.

8.17 Product.h

Go to the documentation of this file.

```
00001
00006 #ifndef PRODUCT_H
00007 #define PRODUCT_H
00008
00009 #include <QString>
00010
00012 enum status {SUCCESS, ERROR};
00013
00020 class Product
00021 {
00022 private:
00023     QString name;
00024     double price;
00025     int quantity;
00026 public:
00027     int productId;
00028
00037     Product(int productId, QString name, double price, int quantity);
00038
00045     status sell(int quantityToSell);
00046
00053     status updatePrice(double newPrice);
00054
00061     status changeQuantity(int quantity);
00062
00063     // Getters
00064     QString getName() const;
00065     double getPrice() const;
00066     int getQuantity() const;
00067 };
00068
00069 #endif // PRODUCT_H
```

8.18 src/Report/Report.cpp File Reference

Source file for the Report class.

```
#include "Report.h"
```

8.18.1 Detailed Description

Source file for the Report class.

Definition in file Report.cpp.

8.19 Report.cpp

Go to the documentation of this file.

```
00001
00006 #include "Report.h"
00007
00008 double Report::operationalCosts = 0;
00009 double Report::netProfit = 0;
00010
00017 Report::Report(double operationalCosts, double netProfit)
00018 {
00019     setOperationalCosts(operationalCosts);
00020     setNetProfit(netProfit);
00021 }
00022
00028 QString Report::generateReport() const
00029 {
00030     return QString("Operational Costs,Net Profit\n%1,%2\n")
00031         .arg(getOperationalCosts())
00032         .arg(getNetProfit());
00033 }
00034
00035 // Setters implementation
00036 void Report::setOperationalCosts(double costs)
00037 {
00038     operationalCosts = costs;
00039 }
00040
00041 void Report::setNetProfit(double profit)
00042 {
00043     netProfit = profit;
00044 }
00045
00046 // Getters implementation
00047 double Report::getOperationalCosts()
00048 {
00049     return operationalCosts;
00050 }
00051
00052 double Report::getNetProfit()
00053 {
00054     return netProfit;
00055 }
```

8.20 src/Report/Report.h File Reference

Header file for the Report class.

```
#include <QString>
```

Classes

- class Report

  Represents a report in the store simulation.
- struct Report::ProductReport

  Represents a report for a single product.

8.20.1 Detailed Description

Header file for the Report class.

Definition in file Report.h.

8.21 Report.h

Go to the documentation of this file.

```
00001
00006 #ifndef REPORT_H
00007 #define REPORT_H
00008
00009 #include <QString>
00010
00017 class Report
00018 {
00019 private:
00020     static double operationalCosts;
00021     static double netProfit;
00022
00023 public:
00024
00031     struct ProductReport
00032     {
00033         QString name;
00034         double price;
00035         int quantity;
00036     };
00037
00044     Report(double operationalCosts, double netProfit);
00045
00052     QString generateReport() const;
00053
00054     // Setters
00055     static void setOperationalCosts(double costs);
00056     static void setNetProfit(double profit);
00057
00058     // Getters
00059     static double getOperationalCosts();
00060     static double getNetProfit();
00061 };
00062
00063 #endif // REPORT_H
```

8.22 src/SalesReport/SalesReport.cpp File Reference

Source file for the SalesReport class.

```
#include "SalesReport.h"
```

8.22.1 Detailed Description

Source file for the SalesReport class.

Definition in file SalesReport.cpp.

8.23 SalesReport.cpp

Go to the documentation of this file.

```
00001
00006 #include "SalesReport.h"
00007
00011 SalesReport::SalesReport(int SalesId, QDateTime time, QList<ProductReport> productList, double
      operationalCosts, double netProfit) : Report(operationalCosts, netProfit)
00012 {
00013     this -> salesId = SalesId;
00014     this -> time = time;
00015     this -> productList = productList;
00016 }
00017
```

```
00021 QString SalesReport::generateReport() const
00022 {
00023     QString report;
00024     report += QString("Sales ID,Time\n%1,%2\n")
00025                   .arg(salesId)
00026                   .arg(time.toString("yyyy-MM-dd hh:mm:ss"));
00027
00028     report += "Product Name,Price,Quantity Sold\n";
00029
00030     for(const ProductReport& product : productList)
00031     {
00032         if(product.quantity > 0)
00033         {
00034             report += QString("%1,%2,%3\n")
00035                           .arg(product.name)
00036                           .arg(product.price)
00037                           .arg(product.quantity);
00038
00039             setNetProfit(getNetProfit() + product.price * product.quantity);
00040         }
00041     }
00042
00043     setNetProfit(getNetProfit()-getOperationalCosts());
00044
00045     report += Report::generateReport();
00046
00047     setNetProfit(getNetProfit()+getOperationalCosts());
00048
00049     return report;
00050 }
```

8.24 src/SalesReport/SalesReport.h File Reference

Header file for the SalesReport class.

```
#include "Report/Report.h"
#include <QDateTime>
#include <QList>
```

Classes

- class SalesReport

  The SalesReport class extends the Report class to provide a report specifically for sales.

8.24.1 Detailed Description

Header file for the SalesReport class.

Definition in file SalesReport.h.

8.25 SalesReport.h

Go to the documentation of this file.

```
00001
00006 #ifndef SALESREPORT_H
00007 #define SALESREPORT_H
00008
00009 #include "Report/Report.h"
00010 #include <QDateTime>
00011 #include <QList>
00012
```

```
00020 class SalesReport : Report
00021 {
00022 private:
00023     int salesId;
00024     QDateTime time;
00025     QList<ProductReport> productList;
00026
00027 public:
00037     SalesReport(int salesId, QDateTime time, QList<ProductReport> productList, double
      operationalCosts, double netProfit);
00038
00043     QString generateReport() const;
00044 };
00045
00046 #endif // SALESREPORT_H
```

8.26 src/Simulation/Simulation.cpp File Reference

Source file for the Simulation class.

```
#include "Simulation.h"
```

8.26.1 Detailed Description

Source file for the Simulation class.

Definition in file Simulation.cpp.

8.27 Simulation.cpp

Go to the documentation of this file.

```
00001
00006 #include "Simulation.h"
00007
00011 Simulation::Simulation()
00012 {
00013     std::cout « "Preparing simulation" « std::endl;
00014
00015     currentTime = QDateTime::currentDateTime();
00016
00017     QFile settings("settings.csv");
00018
00019     QFile::rename("SimulationReport.csv", "SimulationReportOld.csv");
00020     QFile::remove("SimulationReport.csv");
00021
00022     if(!settings.open(QIODevice::ReadOnly | QIODevice::Text))
00023     {
00024         std::cerr « "Error: Can't open settings file." « std::endl;
00025         return;
00026     }
00027     QTextStream in(&settings);
00028
00029     seed = 100;
00030     int productIdCounter = 1;
00031
00032     while(!in.atEnd())
00033     {
00034         QString line = in.readLine();
00035         QStringList fields = line.split(',');
00036
00037         if(fields.isEmpty())
00038         {
00039             continue;
00040         }
00041
00042         if(fields[0] == "Warehouse")
00043         {
```

```
00044             productIdCounter = 1;
00045             if(fields.size() < 3)
00046             {
00047                 std::cerr « "Error: Incomplete warehouse data." « std::endl;
00048                 continue;
00049             }
00050             std::cout « "Adding warehouse." « std::endl;
00051             QString location = fields[1];
00052             double capacity = fields[2].toDouble();
00053             Warehouses.append(Warehouse(location, capacity));
00054         }
00055         else if(fields[0] == "Product")
00056         {
00057             if(fields.size() < 4)
00058             {
00059                 std::cerr « "Error: Incomplete product data." « std::endl;
00060                 continue;
00061             }
00062             std::cout « "Adding product: ";
00063             Warehouse& selectWarehouse = Warehouses.last();
00064             QString name = fields[1];
00065             double price = fields[2].toDouble();
00066             int quantity = fields[3].toInt();
00067             int productId = productIdCounter++;
00068
00069             if(selectWarehouse.addProduct(name, price, quantity, productId) == SUCCESS)
00070             {
00071                 std::cout « "SUCCESS" « std::endl;
00072             }
00073             else
00074             {
00075                 std::cerr « "ERROR: Unable to add product." « std::endl;
00076             }
00077         }
00078         else if(fields[0] == "Cycles")
00079         {
00080             if(fields.size() < 5)
00081             {
00082                 std::cerr « "Error: Incomplete cycle data." « std::endl;
00083                 continue;
00084             }
00085             std::cout « "Setting currentCycle: ";
00086             int currentCycle = fields[4].toInt();
00087             this -> currentCycle = currentCycle;
00088             std::cout « currentCycle « std::endl;
00089         }
00090         else if(fields[0] == "Seed")
00091         {
00092             if(fields.size() < 6)
00093             {
00094                 std::cerr « "Error: Incomplete seed data." « std::endl;
00095                 continue;
00096             }
00097             std::cout « "Setting seed: ";
00098             int readSeed = fields[5].toInt();
00099             this -> seed = QRandomGenerator::global() -> generate()/readSeed;
00100             std::cout « readSeed « std::endl;
00101         }
00102     }
00103
00104     settings.close();
00105 }
00106
00110 void Simulation::run()
00111 {
00112     std::cout « "Running simulation with " « currentCycle « " cycles." « std::endl;
00113
00114     if(Warehouses.isEmpty())
00115     {
00116         std::cout « "Incorrect settings" « std::endl;
00117         exit(1);
00118     }
00119
00120     std::cout « "Generating cycles." « std::endl;
00121
00122     int cycles = currentCycle;
00123
00124     while(currentCycle > 0)
00125     {
00126         std::cout « "Processing cycle " « cycles - currentCycle « "." « std::endl;
00127         int minProductsAvailable = std::numeric_limits<int>::max();
00128         int totalProducts = 0;
00129
00130         for (Warehouse& warehouse : Warehouses)
00131         {
00132             const QList<Product>& productList = warehouse.getProductList();
00133             if (productList.isEmpty())
```

```
00134             {
00135                 std::cout « "Warning: No products available in warehouse at " «
      warehouse.getLocation().toStdString() « std::endl;
00136                 continue;
00137             }
00138
00139             int numberOfEvents = QRandomGenerator::global()->bounded(productList.size()) + 1;
00140
00141             std::cout « "Number of sale events to be generated for warehouse at " «
      warehouse.getLocation().toStdString() « ": " « numberOfEvents « std::endl;
00142
00143             for (int event = 0; event < numberOfEvents; ++event)
00144             {
00145                 events.append(Event::generateEvent("Sell product", seed));
00146                 std::cout « "\033[33mInfo: Generating event - Sell product\033[0m" « std::endl;
00147             }
00148         }
00149
00150         if (events.isEmpty())
00151         {
00152             std::cout « "No events generated in this cycle." « std::endl;
00153         }
00154         else
00155         {
00156             QList<Warehouse> restoreWarehouse = Warehouses;
00157
00158             if (processEvents() == ERROR && currentCycle != cycles)
00159             {
00160                 Warehouses = restoreWarehouse;
00161             }
00162
00163             QString cycleReport = generateReport();
00164             std::cout « cycleReport.toStdString() « std::endl;
00165             events.clear();
00166         }
00167
00168         std::cout « "Cycle " « cycles - currentCycle « " completed." « std::endl;
00169         currentCycle--;
00170     }
00171     std::cout « "Simulation completed." « std::endl;
00172 }
00173
00174
00178 status Simulation::processEvents()
00179 {
00180     if(conductCycle() == ERROR)
00181     {
00182         std::cout « "Error while processing cycle" « std::endl;
00183         return ERROR;
00184     }
00185     return SUCCESS;
00186 }
00187
00191 status Simulation::conductCycle()
00192 {
00193     qint64 deltaTime = 0;
00194
00195     int successEvents = 0;
00196
00197     for(Event& event : events)
00198     {
00199         if(respondToEvent(event) == ERROR)
00200         {
00201             std::cout « "Error while processing event." « std::endl;
00202         }
00203         else
00204         {
00205             successEvents++;
00206         }
00207
00208         QDateTime Time = event.getTime();
00209
00210         if(Time.msecsTo(currentTime) > deltaTime)
00211         {
00212             deltaTime = Time.msecsTo(currentTime);
00213         }
00214     }
00215
00216     if(successEvents == 0)
00217     {
00218         return ERROR;
00219     }
00220
00221     events.clear();
00222     currentTime = currentTime.addSecs(deltaTime);
00223     return SUCCESS;
00224 }
```

```
00225
00229 status Simulation::respondToEvent(Event& event)
00230 {
00231     int warehouseId = QRandomGenerator::global() -> bounded(Warehouses.size());
00232     Warehouse& warehouse = Warehouses[warehouseId];
00233
00234     if(event.getEventType() == "Sell product")
00235     {
00236         try
00237         {
00238             const QList<Product>& productList = warehouse.getProductList();
00239             if (productList.isEmpty())
00240             {
00241                 std::cerr « "ERROR: No products to sell." « std::endl;
00242                 return ERROR;
00243             }
00244
00245             int productId = QRandomGenerator::global()->bounded(productList.size()) +
      productList.begin()->productId;
00246             std::cout « "Attempting to sell" « productId;
00247             status result = warehouse.sell(1, productId);
00248
00249             if (result == ERROR)
00250             {
00251                 std::cout « "ERROR: Unable to sell product. Product doesn't exists." « std::endl;
00252                 return ERROR;
00253             }
00254         }
00255         catch (const std::exception& e)
00256         {
00257             std::cerr « "General Exception caught: " « e.what() « std::endl;
00258             return ERROR;
00259         }
00260     }
00261     else if (event.getEventType() == "Add product")
00262     {
00263         try
00264         {
00265             const QList<Product>& productList = warehouse.getProductList();
00266             int productId = QRandomGenerator::global()->bounded(productList.size());
00267             const Product& product = productList[productId];
00268
00269             status result;
00270
00271             while(warehouse.checkStatus() != FULLY)
00272             {
00273                 result = warehouse.changeQuantity(product.getQuantity() + 1, productId);
00274             }
00275
00276             if (result == ERROR)
00277             {
00278                 std::cerr « "ERROR: Unable to add product quantity." « std::endl;
00279                 return ERROR;
00280             }
00281         }
00282         catch (const std::exception& e)
00283         {
00284             std::cerr « "General Exception caught: " « e.what() « std::endl;
00285             return ERROR;
00286         }
00287     }
00288     else if (event.getEventType() == "Transfer product")
00289     {
00290         try
00291         {
00292             const QList<Product>& productList = warehouse.getProductList();
00293             int productId = QRandomGenerator::global()->bounded(productList.size());
00294             const Product& productToTransfer = productList[productId];
00295
00296             auto targetWarehouseIt = Warehouses.end();
00297             for (auto it = Warehouses.begin(); it != Warehouses.end(); ++it)
00298             {
00299                 if (it->getLocation() != warehouse.getLocation() && it-> checkStatus() == FULLY)
00300                 {
00301                     targetWarehouseIt = it;
00302                     break;
00303                 }
00304             }
00305
00306             if (targetWarehouseIt != Warehouses.end())
00307             {
00308                 Warehouse& targetWarehouse = *targetWarehouseIt;
00309                 int transferQuantity = productToTransfer.getQuantity();
00310                 warehouse.changeQuantity(warehouse.getQuantity(productId) - transferQuantity,
      productId);
00311                 targetWarehouse.changeQuantity(targetWarehouse.getQuantity(productId) +
      transferQuantity, productId);
```

```
00312             }
00313             else
00314             {
00315                 std::cerr « "ERROR: No target warehouse found for product transfer." « std::endl;
00316                 return ERROR;
00317             }
00318         }
00319         catch (const std::exception& e)
00320         {
00321             std::cerr « "General Exception caught: " « e.what() « std::endl;
00322             return ERROR;
00323         }
00324     }
00325
00326     return SUCCESS;
00327 }
00328
00332 QString Simulation::generateReport()
00333 {
00334     QString csvReport;
00335     int id = 0;
00336
00337     std::cout « "Generating report" « std::endl;
00338
00339     for(Warehouse& warehouse : Warehouses)
00340     {
00341         QList<Report::ProductReport> productNames;
00342         double featureOperationalCosts = 0;
00343         for(const Product& product : warehouse.getProductList())
00344         {
00345             Report::ProductReport productReport;
00346             productReport.name = product.getName();
00347             productReport.price = product.getPrice();
00348             productReport.quantity = product.getQuantity();
00349
00350             featureOperationalCosts += productReport.price + productReport.quantity;
00351
00352             productNames.append(productReport);
00353         }
00354
00355         static int salesId = 0;
00356
00357         WarehouseReport warehouseReport(id++, warehouse.getCurrentCapacity(), productNames,
      Report::getOperationalCosts(), Report::getNetProfit());
00358         SalesReport salesReport(salesId++, currentTime, productNames, Report::getOperationalCosts(),
      Report::getNetProfit());
00359
00360         Report::setOperationalCosts(Report::getOperationalCosts()+featureOperationalCosts);
00361
00362         csvReport.append(warehouseReport.generateReport());
00363         csvReport.append(salesReport.generateReport());
00364     }
00365
00366     // Save the CSV report to a file
00367     QFile csvFile("SimulationReport.csv");
00368
00369     if(csvFile.open(QIODevice::Append | QIODevice::Text))
00370     {
00371         QTextStream out(&csvFile);
00372         out « "\n";
00373         out « csvReport;
00374         csvFile.close();
00375     }
00376     else
00377     {
00378         std::cerr « "Error while trying to write the CSV report to file." « std::endl;
00379     }
00380
00381     return csvReport;
00382 }
```

8.28 src/Simulation/Simulation.h File Reference

Header file for the Simulation class.

```
#include <Warehouse/Warehouse.h>
#include <WarehouseReport/WarehouseReport.h>
#include <SalesReport/SalesReport.h>
#include <Event/Event.h>
```

```
#include <QList>
#include <QDateTime>
#include <QFile>
#include <QRandomGenerator>
#include <iostream>
```

Classes

- class Simulation

  The Simulation class manages the overall department store simulation.

8.28.1 Detailed Description

Header file for the Simulation class.

Declares the Simulation class and its members, which are responsible for managing the overall department store simulation.

Definition in file Simulation.h.

8.29 Simulation.h

Go to the documentation of this file.

```
00001
00008 #ifndef SIMULATION_H
00009 #define SIMULATION_H
00010
00011 #include <Warehouse/Warehouse.h>
00012 #include <WarehouseReport/WarehouseReport.h>
00013 #include <SalesReport/SalesReport.h>
00014 #include <Event/Event.h>
00015 #include <QList>
00016 #include <QDateTime>
00017 #include <QFile>
00018 #include <QRandomGenerator>
00019 #include <iostream>
00020
00028 class Simulation
00029 {
00030 private:
00031     int currentCycle;
00032     int seed;
00033     QList<Event> events;
00034     QDateTime currentTime;
00035     QList<Warehouse> Warehouses;
00036
00037 public:
00043     Simulation();
00044
00051     status conductCycle();
00052
00060     status respondToEvent(Event& event);
00061
00067     void run();
00068
00075     status processEvents();
00076
00083     QString generateReport();
00084 };
00085
00086 #endif
```

8.30 src/Storage/Storage.cpp File Reference

Source file of the Storage class.

```
#include "Storage.h"
```

8.30.1 Detailed Description

Source file of the Storage class.

Definition in file Storage.cpp.

8.31 Storage.cpp

Go to the documentation of this file.

```
00001
00006 #include "Storage.h"
00007
00011 Storage::Storage(int capacity)
00012 {
00013     this -> capacity = capacity;
00014 }
00015
00019 storageStatus Storage::checkCapacity(int totalCapacity) const
00020 {
00021     if (capacity == 0)
00022     {
00023         return EMPTY;
00024     }
00025     else if (capacity > 0 && capacity < totalCapacity)
00026     {
00027         return AVAILABLE;
00028     }
00029     else
00030     {
00031         return FULLY;
00032     }
00033 }
```

8.32 src/Storage/Storage.h File Reference

Header file of the Storage class.

Classes

- class Storage

  The Storage class represents a storage unit with a certain capacity.

Enumerations

- enum storageStatus { EMPTY , AVAILABLE , FULLY }

8.32.1 Detailed Description

Header file of the Storage class.

Definition in file Storage.h.

8.32.2 Enumeration Type Documentation

8.32.2.1 storageStatus

```
enum storageStatus
```

Enum representing the status of the Storage capacity.

Enumerator

EMPTY	
AVAILABLE	
FULLY	

Definition at line 10 of file Storage.h.

8.33 Storage.h

Go to the documentation of this file.

```
00001
00006 #ifndef STORAGE_H
00007 #define STORAGE_H
00008
00010 enum storageStatus {EMPTY, AVAILABLE, FULLY};
00011
00019 class Storage
00020 {
00021 protected:
00022     int capacity;
00023 public:
00029     Storage(int capacity);
00030
00037     storageStatus checkCapacity(int totalCapacity) const;
00038 };
00039
00040 #endif
```

8.34 src/Warehouse/Warehouse.cpp File Reference

Source file of the Warehouse class.

```
#include "Warehouse.h"
```

8.34.1 Detailed Description

Source file of the Warehouse class.

Definition in file Warehouse.cpp.

8.35 Warehouse.cpp

Go to the documentation of this file.

```
00001
00006 #include "Warehouse.h"
00007
00009 int Warehouse::warehouseId = 0;
00010
00014 Warehouse::Warehouse(QString location, double warehouseCapacity) : Storage(0)
00015 {
00016     ++warehouseId;
00017     this->location = location;
00018
00019     if (warehouseCapacity >= 0)
00020     {
00021         this->warehouseCapacity = warehouseCapacity;
00022     }
00023     else
00024     {
00025         this->warehouseCapacity = 0;
00026     }
00027 }
00028
00033 storageStatus Warehouse::checkStatus()
00034 {
00035     if (warehouseCapacity == 0)
00036     {
00037         return EMPTY;
00038     }
00039
00040     int sumOfProductInstances = 0;
00041
00042     for (Product& product : productList)
00043     {
00044         sumOfProductInstances += product.getQuantity();
00045     }
00046
00047     if (sumOfProductInstances == warehouseCapacity)
00048     {
00049         return FULLY;
00050     }
00051
00052     capacity = sumOfProductInstances;
00053
00054     return checkCapacity(warehouseCapacity);
00055 }
00056
00060 status Warehouse::updateStatus(int newCapacity)
00061 {
00062     if (newCapacity < 0)
00063     {
00064         return ERROR;
00065     }
00066
00067     int sumOfProductInstances = capacity;
00068
00069     if (newCapacity >= sumOfProductInstances)
00070     {
00071         warehouseCapacity = newCapacity;
00072         capacity = sumOfProductInstances;
00073         return SUCCESS;
00074     }
00075     else
00076     {
00077         return ERROR;
00078     }
00079 }
00080
00084 status Warehouse::addProduct(QString name, double price, int quantity, int productId)
00085 {
00086     int sumOfProductInstances = capacity;
00087
00088     if(sumOfProductInstances + quantity <= warehouseCapacity && quantity >= 0 && price >= 0)
00089     {
00090         for (Product& product : productList)
00091         {
00092             if (product.productId == productId)
00093             {
00094                 return ERROR;
00095             }
00096         }
00097
00098         Product newProduct(productId, name, price, quantity);
00099         productList.append(newProduct);
00100         capacity += quantity;
```

```
00101         return SUCCESS;
00102     }
00103     else
00104     {
00105         return ERROR;
00106     }
00107 }
00108
00112 status Warehouse::updatePrice(double newPrice, int productId)
00113 {
00114     bool productFound = false;
00115
00116     for(Product& product : productList)
00117     {
00118         if(product.productId == productId)
00119         {
00120             return product.updatePrice(newPrice);
00121         }
00122     }
00123
00124     if(productFound == false)
00125     {
00126         return ERROR;
00127     }
00128 }
00129
00133 status Warehouse::changeQuantity(int quantity, int productId)
00134 {
00135     int sumOfProductInstances = capacity;
00136
00137     for(Product& product : productList)
00138     {
00139         if(product.productId == productId)
00140         {
00141             int currentQuantity = product.getQuantity();
00142
00143             if(sumOfProductInstances - currentQuantity + quantity <= warehouseCapacity)
00144             {
00145                 status Status = product.changeQuantity(quantity);
00146
00147                 if(Status == SUCCESS)
00148                 {
00149                     capacity = sumOfProductInstances - currentQuantity + quantity;
00150                     return SUCCESS;
00151                 }
00152                 else
00153                 {
00154                     return ERROR;
00155                 }
00156             }
00157             else
00158             {
00159                 return ERROR;
00160             }
00161         }
00162     }
00163     return ERROR;
00164 }
00165
00169 status Warehouse::sell(int quantityToSell, int productId)
00170 {
00171     for(Product& product : productList)
00172     {
00173         if(product.productId == productId)
00174         {
00175             return product.sell(quantityToSell);
00176         }
00177     }
00178     return ERROR;
00179 }
00180
00184 QString Warehouse::getName(int productId)
00185 {
00186     bool productFound = false;
00187
00188     for(Product& product : productList)
00189     {
00190         if(product.productId == productId)
00191         {
00192             return product.getName();
00193         }
00194     }
00195
00196     if(productFound == false)
00197     {
00198         return "ERROR";
00199     }
```

```
00200 }
00201
00205 double Warehouse::getPrice(int productId)
00206 {
00207     bool productFound = false;
00208
00209     for(Product& product : productList)
00210     {
00211         if(product.productId == productId)
00212         {
00213             return product.getPrice();
00214         }
00215     }
00216
00217     if(productFound == false)
00218     {
00219         return -1;
00220     }
00221 }
00222
00226 int Warehouse::getQuantity(int productId)
00227 {
00228     bool productFound = false;
00229
00230     for(Product& product : productList)
00231     {
00232         if(product.productId == productId)
00233         {
00234             return product.getQuantity();
00235         }
00236     }
00237
00238     if(productFound == false)
00239     {
00240         return -1;
00241     }
00242 }
00243
00247 QString Warehouse::getLocation() const
00248 {
00249     return location;
00250 }
00251
00255 const QList<Product> Warehouse::getProductList() const
00256 {
00257     return productList;
00258 }
00259
00263 double Warehouse::getCurrentCapacity() const
00264 {
00265     return warehouseCapacity;
00266 }
```

8.36 src/Warehouse/Warehouse.h File Reference

Header file of the Warehouse class.

```
#include "Storage/Storage.h"
#include "Product/Product.h"
#include <QList>
```

Classes

- class Warehouse

  The Warehouse class represents a warehouse with storage capacity.

8.36.1 Detailed Description

Header file of the Warehouse class.

Definition in file Warehouse.h.

8.37 Warehouse.h

Go to the documentation of this file.

```
00001
00006 #ifndef WAREHOUSE_H
00007 #define WAREHOUSE_H
00008
00009 #include "Storage/Storage.h"
00010 #include "Product/Product.h"
00011 #include <QList>
00012
00020 class Warehouse : Storage
00021 {
00022 private:
00023     QString location;
00024     QList<Product> productList;
00025     double warehouseCapacity;
00026 public:
00033     Warehouse(QString location, double warehouseCapacity);
00034
00035     static int warehouseId;
00036
00043     storageStatus checkStatus();
00044
00051     status updateStatus(int newCapacity);
00052
00053     // Product operations
00054
00064     status addProduct(QString name, double price, int quantity, int productId);
00065
00073     status updatePrice(double newPrice, int productId);
00074
00082     status changeQuantity(int quantity, int productId);
00083
00091     status sell(int quantityToSell, int productId);
00092
00099     QString getName(int productId);
00100
00107     double getPrice(int productId);
00108
00115     int getQuantity(int productId);
00116
00117
00118
00119     //Getters
00120     QString getLocation() const;
00121     const QList<Product> getProductList() const;
00122     double getCurrentCapacity() const;
00123 };
00124
00125 #endif
```

8.38 src/WarehouseReport/WarehouseReport.cpp File Reference

Source file of the WarehouseReport class.

```
#include "WarehouseReport.h"
```

8.38.1 Detailed Description

Source file of the WarehouseReport class.

Definition in file WarehouseReport.cpp.

8.39 WarehouseReport.cpp

Go to the documentation of this file.

```
00001
00006 #include "WarehouseReport.h"
00007
00011 WarehouseReport::WarehouseReport(int warehouseId, double capacity, QList<ProductReport> productList,
      double operationalCosts, double netProfit) : Report(operationalCosts, netProfit)
00012 {
00013     this -> warehouseId = warehouseId;
00014     this -> capacity = capacity;
00015     this -> productList = productList;
00016 }
00017
00021 QString WarehouseReport::generateReport() const
00022 {
00023     QString report;
00024     report += QString("Warehouse ID,Capacity\n%1,%2\n")
00025                   .arg(warehouseId)
00026                   .arg(capacity);
00027
00028     report += "Product Name,Price,Quantity\n";
00029     for(const ProductReport& product : productList)
00030     {
00031         report += QString("%1,%2,%3\n")
00032                       .arg(product.name)
00033                       .arg(product.price)
00034                       .arg(product.quantity);
00035     }
00036
00037     return report;
00038 }
```

8.40 src/WarehouseReport/WarehouseReport.h File Reference

Header file of the WarehouseReport class.

```
#include <Report/Report.h>
#include <QList>
```

Classes

- class WarehouseReport

  The WarehouseReport class extends the Report class to provide a report specifically for warehouse inventory.

8.40.1 Detailed Description

Header file of the WarehouseReport class.

Definition in file WarehouseReport.h.

8.41 WarehouseReport.h

Go to the documentation of this file.

```
00001
00006 #ifndef WAREHOUSEREPORT_H
00007 #define WAREHOUSEREPORT_H
00008
00009 #include <Report/Report.h>
00010 #include <QList>
00011
00019 class WarehouseReport : Report
00020 {
00021 private:
00022     int warehouseId;
00023     double capacity;
00024     QList<ProductReport> productList;
00025
00026 public:
00036     WarehouseReport(int warehouseId, double capacity, QList<ProductReport> productList, double
      operationalCosts, double netProfit);
00037
00042     QString generateReport() const;
00043 };
00044
00045 #endif
```

8.42 test/Event/EventTest.cpp File Reference

Source file of tests for the Event class.

```
#include <gtest/gtest.h>
#include "Event/Event.h"
```

Functions

- TEST (EventTest, itLives)

  Test to ensure that a Event object can be instantiated.
- TEST (EventTest, generateEventShouldReturnCorrectValue)

  Test to verify that the method generates a valid event.

8.42.1 Detailed Description

Source file of tests for the Event class.

Definition in file EventTest.cpp.

8.42.2 Function Documentation

8.42.2.1 TEST() [1/2]

```
TEST (
            EventTest ,
            generateEventShouldReturnCorrectValue  )
```

Test to verify that the method generates a valid event.

Definition at line 20 of file EventTest.cpp.

8.42.2.2 TEST() [2/2]

```
TEST (
            EventTest ,
            itLives  )
```

Test to ensure that a Event object can be instantiated.

Definition at line 12 of file EventTest.cpp.

8.43 EventTest.cpp

Go to the documentation of this file.

```
00001
00006 #include <gtest/gtest.h>
00007 #include "Event/Event.h"
00008
00012 TEST(EventTest, itLives)
00013 {
00014     Event event("Init", QDateTime::currentDateTime());
00015 }
00016
00020 TEST(EventTest, generateEventShouldReturnCorrectValue)
00021 {
00022     Event event = Event::generateEvent("Init", 100);
00023
00024     EXPECT_FALSE(event.getEventType().isEmpty());
00025     EXPECT_TRUE(event.getTime() > QDateTime::currentDateTime());
00026 }
```

8.44 test/Product/ProductTest.cpp File Reference

Source file of tests for the Product class.

```
#include <gtest/gtest.h>
#include "Product/Product.h"
```

Functions

- TEST (ProductTest, itLives)
  Test to ensure that a Product object can be instantiated.
- TEST (ProductTest, getNameShouldReturnName)
  Test to verify that getName() returns the correct product name.
- TEST (ProductTest, getQuantityShouldReturnInt)
  Test to verify that getQuantity() returns the correct quantity.
- TEST (ProductTest, getPriceShouldReturnDouble)
  Test to verify that getPrice() returns the correct price.
- TEST (ProductTest, sellShouldReturnSuccess)
  Test to verify that sell() returns SUCCESS when selling a valid quantity.
- TEST (ProductTest, sellShouldReturnError)
  Test to verify that sell() returns ERROR when selling an invalid quantity.
- TEST (ProductTest, updatePriceShouldReturnSuccess)
  Test to verify that updatePrice() returns SUCCESS when given a valid price.
- TEST (ProductTest, updatePriceShouldReturnError)
  Test to verify that updatePrice() returns ERROR when given an invalid price.
- TEST (ProductTest, changeQuantityShouldReturnSuccess)
  Test to verify that changeQuantity() returns SUCCESS when given a valid quantity.
- TEST (ProductTest, changeQuantityShouldReturnError)
  Test to verify that changeQuantity() returns ERROR when given an invalid quantity.

8.44.1 Detailed Description

Source file of tests for the Product class.

Definition in file ProductTest.cpp.

8.44.2 Function Documentation

8.44.2.1 TEST() [1/10]

```
TEST (
            ProductTest ,
            changeQuantityShouldReturnError  )
```

Test to verify that changeQuantity() returns ERROR when given an invalid quantity.

Definition at line 110 of file ProductTest.cpp.

8.44.2.2 TEST() [2/10]

```
TEST (
            ProductTest ,
            changeQuantityShouldReturnSuccess  )
```

Test to verify that changeQuantity() returns SUCCESS when given a valid quantity.

Definition at line 99 of file ProductTest.cpp.

8.44.2.3 TEST() [3/10]

```
TEST (
            ProductTest ,
            getNameShouldReturnName  )
```

Test to verify that getName() returns the correct product name.

Definition at line 20 of file ProductTest.cpp.

8.44.2.4 TEST() [4/10]

```
TEST (
            ProductTest ,
            getPriceShouldReturnDouble  )
```

Test to verify that getPrice() returns the correct price.

Definition at line 38 of file ProductTest.cpp.

8.44.2.5 TEST() [5/10]

```
TEST (
            ProductTest ,
            getQuantityShouldReturnInt  )
```

Test to verify that getQuantity() returns the correct quantity.

Definition at line 29 of file ProductTest.cpp.

8.44.2.6 TEST() [6/10]

```
TEST (
            ProductTest ,
            itLives  )
```

Test to ensure that a Product object can be instantiated.

Definition at line 12 of file ProductTest.cpp.

8.44.2.7 TEST() [7/10]

```
TEST (
            ProductTest ,
            sellShouldReturnError  )
```

Test to verify that sell() returns ERROR when selling an invalid quantity.

Definition at line 60 of file ProductTest.cpp.

8.44.2.8 TEST() [8/10]

```
TEST (
            ProductTest ,
            sellShouldReturnSuccess  )
```

Test to verify that sell() returns SUCCESS when selling a valid quantity.

Definition at line 47 of file ProductTest.cpp.

8.44.2.9 TEST() [9/10]

```
TEST (
            ProductTest ,
            updatePriceShouldReturnError  )
```

Test to verify that updatePrice() returns ERROR when given an invalid price.

Definition at line 86 of file ProductTest.cpp.

8.44.2.10 TEST() [10/10]

```
TEST (
            ProductTest ,
            updatePriceShouldReturnSuccess  )
```

Test to verify that updatePrice() returns SUCCESS when given a valid price.

Definition at line 73 of file ProductTest.cpp.

8.45 ProductTest.cpp

Go to the documentation of this file.

```
00001
00006 #include <gtest/gtest.h>
00007 #include "Product/Product.h"
00008
00012 TEST(ProductTest, itLives)
00013 {
00014     Product(0, QString::fromStdString("Lorem Ipsum"), 1.00, 1);
00015 }
00016
00020 TEST(ProductTest, getNameShouldReturnName)
00021 {
00022     Product product = Product(0, QString::fromStdString("Lorem Ipsum"), 1.00, 1);
00023     EXPECT_EQ(product.getName(),QString::fromStdString("Lorem Ipsum"));
00024 }
00025
00029 TEST(ProductTest, getQuantityShouldReturnInt)
00030 {
00031     Product product = Product(0, QString::fromStdString("Lorem Ipsum"), 1.00, 1);
00032     EXPECT_EQ(product.getQuantity(),1);
00033 }
00034
00038 TEST(ProductTest, getPriceShouldReturnDouble)
00039 {
00040     Product product = Product(0, QString::fromStdString("Lorem Ipsum"), 1.00, 1);
00041     EXPECT_EQ(product.getPrice(),1.00);
00042 }
00043
00047 TEST(ProductTest, sellShouldReturnSuccess)
00048 {
00049     Product product = Product(0, QString::fromStdString("Lorem Ipsum"), 1.00, 1);
00050     Product product2 = Product(1, QString::fromStdString("Ipsum Lorem"), 23.99, 56);
00051     EXPECT_EQ(product.sell(1),SUCCESS);
00052     EXPECT_EQ(product2.sell(1),SUCCESS);
00053     EXPECT_EQ(product2.sell(53),SUCCESS);
00054     EXPECT_EQ(product2.sell(2),SUCCESS);
00055 }
00056
00060 TEST(ProductTest, sellShouldReturnError)
00061 {
00062     Product product = Product(0, QString::fromStdString("Lorem Ipsum"), 1.00, 0);
00063     Product product2 = Product(1, QString::fromStdString("Ipsum Lorem"), 23.99, 1);
00064     EXPECT_EQ(product.sell(1),ERROR);
00065     EXPECT_EQ(product.sell(-1),ERROR);
00066     EXPECT_EQ(product2.sell(2),ERROR);
00067     EXPECT_EQ(product2.sell(-1),ERROR);
00068 }
00069
00073 TEST(ProductTest, updatePriceShouldReturnSuccess)
00074 {
00075     Product product = Product(0, QString::fromStdString("Lorem Ipsum"), 1.00, 0);
00076     EXPECT_EQ(product.updatePrice(1.00),SUCCESS);
00077     EXPECT_EQ(product.updatePrice(59.99),SUCCESS);
00078     EXPECT_EQ(product.updatePrice(545454),SUCCESS);
00079     EXPECT_EQ(product.updatePrice(1),SUCCESS);
00080     EXPECT_EQ(product.updatePrice(0.01),SUCCESS);
00081 }
00082
00086 TEST(ProductTest, updatePriceShouldReturnError)
00087 {
00088     Product product = Product(0, QString::fromStdString("Lorem Ipsum"), 1.00, 0);
00089     EXPECT_EQ(product.updatePrice(-1.00),ERROR);
00090     EXPECT_EQ(product.updatePrice(-59.99),ERROR);
00091     EXPECT_EQ(product.updatePrice(-545454),ERROR);
```

```
00092     EXPECT_EQ(product.updatePrice(-1),ERROR);
00093     EXPECT_EQ(product.updatePrice(-0.01),ERROR);
00094 }
00095
00099 TEST(ProductTest, changeQuantityShouldReturnSuccess)
00100 {
00101     Product product = Product(0, QString::fromStdString("Lorem Ipsum"), 1.00, 3);
00102     EXPECT_EQ(product.changeQuantity(1),SUCCESS);
00103     EXPECT_EQ(product.changeQuantity(59),SUCCESS);
00104     EXPECT_EQ(product.changeQuantity(545454),SUCCESS);
00105 }
00106
00110 TEST(ProductTest, changeQuantityShouldReturnError)
00111 {
00112     Product product = Product(0, QString::fromStdString("Lorem Ipsum"), 1.00, 3);
00113     EXPECT_EQ(product.changeQuantity(-1),ERROR);
00114     EXPECT_EQ(product.changeQuantity(-59),ERROR);
00115     EXPECT_EQ(product.changeQuantity(-545454),ERROR);
00116 }
```

8.46 test/Report/ReportTest.cpp File Reference

Source file of tests for the Report class.

```
#include <gtest/gtest.h>
#include "Report/Report.h"
```

Functions

- TEST (ReportTest, generateReportShouldReturnNonEmptyString)

  Test to ensure that the generateReport method does not return an empty string.
- TEST (ReportTest, GenerateReportShouldReturnExpectedFormat)

  Test to ensure that the generateReport method returns a string in the expected format.

8.46.1 Detailed Description

Source file of tests for the Report class.

Definition in file ReportTest.cpp.

8.46.2 Function Documentation

8.46.2.1 TEST() [1/2]

```
TEST (
            ReportTest ,
            GenerateReportShouldReturnExpectedFormat  )
```

Test to ensure that the generateReport method returns a string in the expected format.

Definition at line 24 of file ReportTest.cpp.

8.46.2.2 TEST() [2/2]

```
TEST (
            ReportTest ,
            generateReportShouldReturnNonEmptyString  )
```

Test to ensure that the generateReport method does not return an empty string.

Definition at line 12 of file ReportTest.cpp.

8.47 ReportTest.cpp

Go to the documentation of this file.

```
00001
00006 #include <gtest/gtest.h>
00007 #include "Report/Report.h"
00008
00012 TEST(ReportTest, generateReportShouldReturnNonEmptyString)
00013 {
00014     Report report(1000.0, 500.0);
00015
00016     QString reportString = report.generateReport();
00017
00018     EXPECT_FALSE(reportString.isEmpty());
00019 }
00020
00024 TEST(ReportTest, GenerateReportShouldReturnExpectedFormat)
00025 {
00026     Report report(1000.0, 500.0);
00027
00028     QString reportString = report.generateReport();
00029
00030     EXPECT_TRUE(reportString.contains("Operational Costs"));
00031     EXPECT_TRUE(reportString.contains("Net Profit"));
00032 }
```

8.48 test/SalesReport/SalesReportTest.cpp File Reference

Source file of tests for the SalesReport class.

```
#include <gtest/gtest.h>
#include "SalesReport/SalesReport.h"
#include <QDateTime>
```

Functions

- TEST (SalesReportTest, itLives)

  Test case to ensure that a SalesReport object can be instantiated.
- TEST (SalesReportTest, generateReportShouldReturnCorrectValue)

  Test case to verify that the SalesReport::generateReport method returns the correct CSV format.

8.48.1 Detailed Description

Source file of tests for the SalesReport class.

Definition in file SalesReportTest.cpp.

8.48.2 Function Documentation

8.48.2.1 TEST() [1/2]

```
TEST (
            SalesReportTest ,
            generateReportShouldReturnCorrectValue  )
```

Test case to verify that the SalesReport::generateReport method returns the correct CSV format.

Definition at line 27 of file SalesReportTest.cpp.

8.48.2.2 TEST() [2/2]

```
TEST (
            SalesReportTest ,
            itLives  )
```

Test case to ensure that a SalesReport object can be instantiated.

Definition at line 13 of file SalesReportTest.cpp.

8.49 SalesReportTest.cpp

Go to the documentation of this file.

```
00001
00006 #include <gtest/gtest.h>
00007 #include "SalesReport/SalesReport.h"
00008 #include <QDateTime>
00009
00013 TEST(SalesReportTest, itLives)
00014 {
00015     QDateTime time = QDateTime::currentDateTime();
00016     QList<Report::ProductReport> productList;
00017     double operationalCosts = 5000.0;
00018     double netProfit = 10000.0;
00019     int salesId = 1;
00020
00021     SalesReport salesReport(salesId, time, productList, operationalCosts, netProfit);
00022 }
00023
00027 TEST(SalesReportTest, generateReportShouldReturnCorrectValue)
00028 {
00029     QDateTime time = QDateTime::currentDateTime();
00030     QList<Report::ProductReport> productList =
00031         {
00032             {"Product1", 10.0, 5},
00033             {"Product2", 20.0, 3}
00034         };
00035
00036     double operationalCosts = 5000.0;
00037     double netProfit = 10000.0;
00038     int salesId = 1;
00039
00040     SalesReport report(salesId, time, productList, operationalCosts, netProfit);
00041     QString generatedCSV = report.generateReport();
00042
00043     QString expectedCSVStart = QString("Sales
      ID,Time\n%1,%2\n").arg(salesId).arg(time.toString("yyyy-MM-dd hh:mm:ss"));
00044     ASSERT_TRUE(generatedCSV.startsWith(expectedCSVStart));
00045 }
```

8.50 test/Simulation/SimulationTest.cpp File Reference

Source file of tests for the Simulation class.

```
#include <gtest/gtest.h>
#include "Simulation/Simulation.h"
```

Functions

- TEST (SimulationTest, itLives)

  Test to ensure that the constructor initializes the current time correctly.
- TEST (SimulationTest, processEvents)

  Test to validate the processEvents method.
- TEST (SimulationTest, conductCycle)

  Test to check if the conductCycle method processes the simulation cycle successfully.
- TEST (SimulationTest, generateReport)

  Test to ensure that the generateReport method produces a report.

8.50.1 Detailed Description

Source file of tests for the Simulation class.

Definition in file SimulationTest.cpp.

8.50.2 Function Documentation

8.50.2.1 TEST() [1/4]

```
TEST (
            SimulationTest ,
            conductCycle  )
```

Test to check if the conductCycle method processes the simulation cycle successfully.

Definition at line 29 of file SimulationTest.cpp.

8.50.2.2 TEST() [2/4]

```
TEST (
            SimulationTest ,
            generateReport  )
```

Test to ensure that the generateReport method produces a report.

Definition at line 39 of file SimulationTest.cpp.

8.50.2.3 TEST() [3/4]

```
TEST (
            SimulationTest ,
            itLives  )
```

Test to ensure that the constructor initializes the current time correctly.

Definition at line 12 of file SimulationTest.cpp.

8.50.2.4 TEST() [4/4]

```
TEST (
            SimulationTest ,
            processEvents  )
```

Test to validate the processEvents method.

Definition at line 20 of file SimulationTest.cpp.

8.51 SimulationTest.cpp

Go to the documentation of this file.

```
00001
00006 #include <gtest/gtest.h>
00007 #include "Simulation/Simulation.h"
00008
00012 TEST(SimulationTest, itLives)
00013 {
00014     Simulation simulation = Simulation();
00015 }
00016
00020 TEST(SimulationTest, processEvents)
00021 {
00022     Simulation simulation;
00023     simulation.processEvents();
00024 }
00025
00029 TEST(SimulationTest, conductCycle)
00030 {
00031     Simulation simulation;
00032     status result = simulation.conductCycle();
00033     ASSERT_EQ(result, SUCCESS);
00034 }
00035
00039 TEST(SimulationTest, generateReport)
00040 {
00041     Simulation simulation;
00042     QString report = simulation.generateReport();
00043     ASSERT_TRUE(report.contains("Warehouse ID,"));
00044     ASSERT_TRUE(report.contains("Capacity"));
00045     ASSERT_TRUE(report.contains("Product Name,Price,Quantity"));
00046 }
00047
```

8.52 test/Storage/StorageTest.cpp File Reference

Source file of tests for the Storage class.

```
#include <gtest/gtest.h>
#include "Storage/Storage.h"
```

Functions

- TEST (StorageTest, itLives)

  Test to ensure that a Storage object can be instantiated.
- TEST (StorageTest, checkCapacityShouldReturnEmpty)

  Test to verify that a new Storage is EMPTY when initialized with zero capacity.
- TEST (StorageTest, checkCapacityShouldReturnAvailable)

  Test to verify that Storage is AVAILABLE when initialized with capacity less than total capacity.
- TEST (StorageTest, checkCapacityShouldReturnFully)

  Test to verify that Storage is FULLY when initialized with capacity equal to or greater than total capacity.

8.52.1 Detailed Description

Source file of tests for the Storage class.

Definition in file StorageTest.cpp.

8.52.2 Function Documentation

8.52.2.1 TEST() [1/4]

```
TEST (
            StorageTest ,
            checkCapacityShouldReturnAvailable  )
```

Test to verify that Storage is AVAILABLE when initialized with capacity less than total capacity.

Definition at line 29 of file StorageTest.cpp.

8.52.2.2 TEST() [2/4]

```
TEST (
            StorageTest ,
            checkCapacityShouldReturnEmpty  )
```

Test to verify that a new Storage is EMPTY when initialized with zero capacity.

Definition at line 20 of file StorageTest.cpp.

8.52.2.3 TEST() [3/4]

```
TEST (
            StorageTest ,
            checkCapacityShouldReturnFully  )
```

Test to verify that Storage is FULLY when initialized with capacity equal to or greater than total capacity.

Definition at line 38 of file StorageTest.cpp.

8.52.2.4 TEST() [4/4]

```
TEST (
             StorageTest ,
             itLives  )
```

Test to ensure that a Storage object can be instantiated.

Definition at line 12 of file StorageTest.cpp.

8.53 StorageTest.cpp

Go to the documentation of this file.

```
00001
00006 #include <gtest/gtest.h>
00007 #include "Storage/Storage.h"
00008
00012 TEST(StorageTest, itLives)
00013 {
00014     Storage(0);
00015 }
00016
00020 TEST(StorageTest, checkCapacityShouldReturnEmpty)
00021 {
00022     Storage storage(0);
00023     EXPECT_EQ(storage.checkCapacity(100), EMPTY);
00024 }
00025
00029 TEST(StorageTest, checkCapacityShouldReturnAvailable)
00030 {
00031     Storage storage(10);
00032     EXPECT_EQ(storage.checkCapacity(100), AVAILABLE);
00033 }
00034
00038 TEST(StorageTest, checkCapacityShouldReturnFully)
00039 {
00040     Storage storage(100);
00041     EXPECT_EQ(storage.checkCapacity(100), FULLY);
00042
00043     Storage storage2(150);
00044     EXPECT_EQ(storage2.checkCapacity(100), FULLY);
00045 }
```

8.54 test/Warehouse/WarehouseTest.cpp File Reference

Source file of tests for the Warehouse class.

```
#include <gtest/gtest.h>
#include "Product/Product.h"
#include "Warehouse/Warehouse.h"
```

Functions

- TEST (WarehouseTest, itLives)

  Test to ensure that a Warehouse object can be instantiated.
- TEST (WarehouseTest, checkStatusShouldReturnEmpty)

  Test to check if the warehouse status is EMPTY when capacity is zero or after products are removed.
- TEST (WarehouseTest, checkStatusShouldReturnFully)

  Test to check if the warehouse status is FULLY when capacity is full or exceeded.

- TEST (WarehouseTest, checkStatusShouldReturnAvailable)

  Test to check if the warehouse status is AVAILABLE when capacity is not full.
- TEST (WarehouseTest, updateStatusShouldReturnSuccess)

  Test to verify that updateStatus successfully updates the warehouse's capacity.
- TEST (WarehouseTest, updateStatusShouldReturnError)

  Test to verify that updateStatus returns an error when given a negative capacity.
- TEST (WarehouseTest, changeQuantityShouldReturnSuccess)

  Test to verify that changeQuantity successfully changes the quantity of a product.
- TEST (WarehouseTest, changeQuantityShouldReturnError)

  Test to verify that changeQuantity returns an error when given a negative quantity.
- TEST (WarehouseTest, sellShouldReturnSuccess)

  Test to verify that sell successfully sells the product and decreases the quantity.
- TEST (WarehouseTest, sellShouldReturnError)

  Test to verify that sell returns an error when trying to sell more than available quantity.
- TEST (WarehouseTest, updatePriceShouldReturnSuccess)

  Test to verify that updatePrice successfully updates the price of a product.
- TEST (WarehouseTest, updatePriceShouldReturnError)

  Test to verify that updatePrice returns an error when given a negative price.
- TEST (WarehouseTest, addProductShouldReturnSuccess)

  Test to verify that addProduct successfully adds a product to the warehouse.
- TEST (WarehouseTest, addProductShouldReturnError)

  Test to verify that addProduct returns an error when given a negative price or quantity.
- TEST (WarehouseTest, getLocationShouldReturnCorrectValue)

  Test to verify that getLocation returns the correct location of the warehouse.
- TEST (WarehouseTest, getProductListShouldReturnCorrectValue)

  Test to verify that getProductList returns the correct list of products.
- TEST (WarehouseTest, getCurrentCapacityShouldReturnCorrectValue)

  Test to verify that getCurrentCapacity returns the correct current capacity of the warehouse.
- TEST (WarehouseTest, getNameShouldReturnCorrectValue)

  Test to verify that getName returns the correct name of a product by its ID.
- TEST (WarehouseTest, getPriceShouldReturnCorrectValue)

  Test to verify that getPrice returns the correct price of a product by its ID.
- TEST (WarehouseTest, getQuantityShouldReturnCorrectValue)

  Test to verify that getQuantity returns the correct quantity of a product by its ID.

Variables

- const QString testLocation = "Test Location"
- const double initialCapacity = 100.0
- const QString productName = "Test Product"
- const double productPrice = 10.0
- const int productQuantity = 20
- const int productId = 1

8.54.1 Detailed Description

Source file of tests for the Warehouse class.

Definition in file WarehouseTest.cpp.

8.54.2 Function Documentation

8.54.2.1 TEST() [1/20]

```
TEST (
            WarehouseTest ,
            addProductShouldReturnError  )
```

Test to verify that addProduct returns an error when given a negative price or quantity.

Definition at line 171 of file WarehouseTest.cpp.

8.54.2.2 TEST() [2/20]

```
TEST (
            WarehouseTest ,
            addProductShouldReturnSuccess  )
```

Test to verify that addProduct successfully adds a product to the warehouse.

Definition at line 161 of file WarehouseTest.cpp.

8.54.2.3 TEST() [3/20]

```
TEST (
            WarehouseTest ,
            changeQuantityShouldReturnError  )
```

Test to verify that changeQuantity returns an error when given a negative quantity.

Definition at line 109 of file WarehouseTest.cpp.

8.54.2.4 TEST() [4/20]

```
TEST (
            WarehouseTest ,
            changeQuantityShouldReturnSuccess  )
```

Test to verify that changeQuantity successfully changes the quantity of a product.

Definition at line 98 of file WarehouseTest.cpp.

8.54.2.5 TEST() [5/20]

```
TEST (
            WarehouseTest ,
            checkStatusShouldReturnAvailable  )
```

Test to check if the warehouse status is AVAILABLE when capacity is not full.

Definition at line 50 of file WarehouseTest.cpp.

8.54.2.6 TEST() [6/20]

```
TEST (
            WarehouseTest ,
            checkStatusShouldReturnEmpty  )
```

Test to check if the warehouse status is EMPTY when capacity is zero or after products are removed.

Definition at line 21 of file WarehouseTest.cpp.

8.54.2.7 TEST() [7/20]

```
TEST (
            WarehouseTest ,
            checkStatusShouldReturnFully  )
```

Test to check if the warehouse status is FULLY when capacity is full or exceeded.

Definition at line 35 of file WarehouseTest.cpp.

8.54.2.8 TEST() [8/20]

```
TEST (
            WarehouseTest ,
            getCurrentCapacityShouldReturnCorrectValue  )
```

Test to verify that getCurrentCapacity returns the correct current capacity of the warehouse.

Definition at line 205 of file WarehouseTest.cpp.

8.54.2.9 TEST() [9/20]

```
TEST (
            WarehouseTest ,
            getLocationShouldReturnCorrectValue  )
```

Test to verify that getLocation returns the correct location of the warehouse.

Definition at line 180 of file WarehouseTest.cpp.

8.54.2.10 TEST() [10/20]

```
TEST (
            WarehouseTest ,
            getNameShouldReturnCorrectValue  )
```

Test to verify that getName returns the correct name of a product by its ID.

Definition at line 214 of file WarehouseTest.cpp.

8.54.2.11 TEST() [11/20]

```
TEST (
            WarehouseTest ,
            getPriceShouldReturnCorrectValue  )
```

Test to verify that getPrice returns the correct price of a product by its ID.

Definition at line 224 of file WarehouseTest.cpp.

8.54.2.12 TEST() [12/20]

```
TEST (
            WarehouseTest ,
            getProductListShouldReturnCorrectValue  )
```

Test to verify that getProductList returns the correct list of products.

Definition at line 189 of file WarehouseTest.cpp.

8.54.2.13 TEST() [13/20]

```
TEST (
            WarehouseTest ,
            getQuantityShouldReturnCorrectValue  )
```

Test to verify that getQuantity returns the correct quantity of a product by its ID.

Definition at line 234 of file WarehouseTest.cpp.

8.54.2.14 TEST() [14/20]

```
TEST (
            WarehouseTest ,
            itLives  )
```

Test to ensure that a Warehouse object can be instantiated.

Definition at line 13 of file WarehouseTest.cpp.

8.54.2.15 TEST() [15/20]

```
TEST (
            WarehouseTest ,
            sellShouldReturnError  )
```

Test to verify that sell returns an error when trying to sell more than available quantity.

Definition at line 130 of file WarehouseTest.cpp.

8.54.2.16 TEST() [16/20]

```
TEST (
            WarehouseTest ,
            sellShouldReturnSuccess  )
```

Test to verify that sell successfully sells the product and decreases the quantity.

Definition at line 119 of file WarehouseTest.cpp.

8.54.2.17 TEST() [17/20]

```
TEST (
            WarehouseTest ,
            updatePriceShouldReturnError  )
```

Test to verify that updatePrice returns an error when given a negative price.

Definition at line 151 of file WarehouseTest.cpp.

8.54.2.18 TEST() [18/20]

```
TEST (
            WarehouseTest ,
            updatePriceShouldReturnSuccess  )
```

Test to verify that updatePrice successfully updates the price of a product.

Definition at line 140 of file WarehouseTest.cpp.

8.54.2.19 TEST() [19/20]

```
TEST (
            WarehouseTest ,
            updateStatusShouldReturnError  )
```

Test to verify that updateStatus returns an error when given a negative capacity.

Definition at line 79 of file WarehouseTest.cpp.

8.54.2.20 TEST() [20/20]

```
TEST (
            WarehouseTest ,
            updateStatusShouldReturnSuccess  )
```

Test to verify that updateStatus successfully updates the warehouse's capacity.

Definition at line 65 of file WarehouseTest.cpp.

8.54.3 Variable Documentation

8.54.3.1 initialCapacity

```
const double initialCapacity = 100.0
```

Definition at line 89 of file WarehouseTest.cpp.

8.54.3.2 productId

```
const int productId = 1
```

Definition at line 93 of file WarehouseTest.cpp.

8.54.3.3 productName

```
const QString productName = "Test Product"
```

Definition at line 90 of file WarehouseTest.cpp.

8.54.3.4 productPrice

```
const double productPrice = 10.0
```

Definition at line 91 of file WarehouseTest.cpp.

8.54.3.5 productQuantity

```
const int productQuantity = 20
```

Definition at line 92 of file WarehouseTest.cpp.

8.54.3.6 testLocation

```
const QString testLocation = "Test Location"
```

Definition at line 88 of file WarehouseTest.cpp.

8.55 WarehouseTest.cpp

Go to the documentation of this file.

```
00001
00006 #include <gtest/gtest.h>
00007 #include "Product/Product.h"
00008 #include "Warehouse/Warehouse.h"
00009
00013 TEST(WarehouseTest, itLives)
00014 {
00015     Warehouse warehouse = Warehouse(QString::fromStdString("Lorem Ipsum"), 532);
00016 }
00017
00021 TEST(WarehouseTest, checkStatusShouldReturnEmpty)
00022 {
00023     Warehouse warehouse = Warehouse(QString::fromStdString("Lorem Ipsum"), 0);
00024     EXPECT_EQ(warehouse.checkStatus(), EMPTY);
00025
00026     warehouse.updateStatus(10);
00027     warehouse.addProduct(QString::fromStdString("Lorem Ipsum"), 99.99, 10, 1);
00028     warehouse.changeQuantity(0, 1);
00029     EXPECT_EQ(warehouse.checkStatus(), EMPTY);
00030 }
00031
00035 TEST(WarehouseTest, checkStatusShouldReturnFully)
00036 {
00037     Warehouse warehouse = Warehouse(QString::fromStdString("Lorem Ipsum"), 1);
00038     warehouse.addProduct(QString::fromStdString("Lorem Ipsum"), 100, 1, 1);
00039     EXPECT_EQ(warehouse.checkStatus(), FULLY);
00040
00041     warehouse.updateStatus(10);
00042     warehouse.addProduct(QString::fromStdString("Lorem Ipsum"), 100, 9, 2);
00043     warehouse.changeQuantity(2, 1);
00044     EXPECT_EQ(warehouse.checkStatus(), FULLY);
00045 }
00046
00050 TEST(WarehouseTest, checkStatusShouldReturnAvailable)
00051 {
00052     Warehouse warehouse = Warehouse(QString::fromStdString("Lorem Ipsum"), 2);
00053     warehouse.addProduct(QString::fromStdString("Lorem Ipsum"), 99.99, 1, 0);
00054     EXPECT_EQ(warehouse.checkStatus(), AVAILABLE);
00055
00056     warehouse.updateStatus(10);
00057     warehouse.addProduct(QString::fromStdString("Lorem Ipsum"), 99.99, 8, 1);
00058     warehouse.changeQuantity(2, 1);
00059     EXPECT_EQ(warehouse.checkStatus(), AVAILABLE);
00060 }
00061
00065 TEST(WarehouseTest, updateStatusShouldReturnSuccess)
00066 {
00067     Warehouse warehouse = Warehouse(QString::fromStdString("Lorem Ipsum"), 0);
00068     EXPECT_EQ(warehouse.updateStatus(10), SUCCESS);
00069     EXPECT_EQ(warehouse.updateStatus(0), SUCCESS);
00070     EXPECT_EQ(warehouse.updateStatus(567.97), SUCCESS);
00071
00072     warehouse.addProduct(QString::fromStdString("Lorem Ipsum"), 99.99, 5, 1);
00073     EXPECT_EQ(warehouse.updateStatus(1), ERROR);
00074 }
00075
00079 TEST(WarehouseTest, updateStatusShouldReturnError)
00080 {
00081     Warehouse warehouse = Warehouse(QString::fromStdString("Lorem Ipsum"), 567);
00082     EXPECT_EQ(warehouse.updateStatus(-1), ERROR);
00083
00084     warehouse.addProduct(QString::fromStdString("Lorem Ipsum"), 100, 5, 1);
00085     EXPECT_EQ(warehouse.updateStatus(1), ERROR);
00086 }
00087
00088 const QString testLocation = "Test Location";
00089 const double initialCapacity = 100.0;
00090 const QString productName = "Test Product";
00091 const double productPrice = 10.0;
00092 const int productQuantity = 20;
00093 const int productId = 1;
00094
00098 TEST(WarehouseTest, changeQuantityShouldReturnSuccess)
00099 {
00100     Warehouse warehouse(testLocation, initialCapacity);
00101     warehouse.addProduct(productName, productPrice, productQuantity, productId);
00102     EXPECT_EQ(warehouse.changeQuantity(30, productId), SUCCESS);
00103     EXPECT_EQ(warehouse.getQuantity(productId), 30);
00104 }
00105
00109 TEST(WarehouseTest, changeQuantityShouldReturnError)
00110 {
```

```
00111     Warehouse warehouse(testLocation, initialCapacity);
00112     warehouse.addProduct(productName, productPrice, productQuantity, productId);
00113     EXPECT_EQ(warehouse.changeQuantity(-5, productId), ERROR);
00114 }
00115
00119 TEST(WarehouseTest, sellShouldReturnSuccess)
00120 {
00121     Warehouse warehouse(testLocation, initialCapacity);
00122     warehouse.addProduct(productName, productPrice, productQuantity, productId);
00123     EXPECT_EQ(warehouse.sell(5, productId), SUCCESS);
00124     EXPECT_EQ(warehouse.getQuantity(productId), productQuantity - 5);
00125 }
00126
00130 TEST(WarehouseTest, sellShouldReturnError)
00131 {
00132     Warehouse warehouse(testLocation, initialCapacity);
00133     warehouse.addProduct(productName, productPrice, productQuantity, productId);
00134     EXPECT_EQ(warehouse.sell(productQuantity + 1, productId), ERROR);
00135 }
00136
00140 TEST(WarehouseTest, updatePriceShouldReturnSuccess)
00141 {
00142     Warehouse warehouse(testLocation, initialCapacity);
00143     warehouse.addProduct(productName, productPrice, productQuantity, productId);
00144     EXPECT_EQ(warehouse.updatePrice(15.0, productId), SUCCESS);
00145     EXPECT_EQ(warehouse.getPrice(productId), 15.0);
00146 }
00147
00151 TEST(WarehouseTest, updatePriceShouldReturnError)
00152 {
00153     Warehouse warehouse(testLocation, initialCapacity);
00154     warehouse.addProduct(productName, productPrice, productQuantity, productId);
00155     EXPECT_EQ(warehouse.updatePrice(-1.0, productId), ERROR);
00156 }
00157
00161 TEST(WarehouseTest, addProductShouldReturnSuccess)
00162 {
00163     Warehouse warehouse(testLocation, initialCapacity);
00164     EXPECT_EQ(warehouse.addProduct(productName, productPrice, productQuantity, productId), SUCCESS);
00165     EXPECT_EQ(warehouse.getProductList().size(), 1);
00166 }
00167
00171 TEST(WarehouseTest, addProductShouldReturnError)
00172 {
00173     Warehouse warehouse(testLocation, initialCapacity);
00174     EXPECT_EQ(warehouse.addProduct(productName, -productPrice, productQuantity, productId), ERROR);
00175 }
00176
00180 TEST(WarehouseTest, getLocationShouldReturnCorrectValue)
00181 {
00182     Warehouse warehouse(testLocation, initialCapacity);
00183     EXPECT_EQ(warehouse.getLocation(), testLocation);
00184 }
00185
00189 TEST(WarehouseTest, getProductListShouldReturnCorrectValue)
00190 {
00191     Warehouse warehouse(testLocation, initialCapacity);
00192     warehouse.addProduct(productName, productPrice, productQuantity, productId);
00193
00194     QList<Product> productList = warehouse.getProductList();
00195
00196     ASSERT_EQ(productList.size(), 1);
00197     EXPECT_EQ(productList[0].getName(), productName);
00198     EXPECT_EQ(productList[0].getPrice(), productPrice);
00199     EXPECT_EQ(productList[0].getQuantity(), productQuantity);
00200 }
00201
00205 TEST(WarehouseTest, getCurrentCapacityShouldReturnCorrectValue)
00206 {
00207     Warehouse warehouse(testLocation, initialCapacity);
00208     EXPECT_EQ(warehouse.getCurrentCapacity(), initialCapacity);
00209 }
00210
00214 TEST(WarehouseTest, getNameShouldReturnCorrectValue)
00215 {
00216     Warehouse warehouse(testLocation, initialCapacity);
00217     warehouse.addProduct(productName, productPrice, productQuantity, productId);
00218     EXPECT_EQ(warehouse.getName(productId), productName);
00219 }
00220
00224 TEST(WarehouseTest, getPriceShouldReturnCorrectValue)
00225 {
00226     Warehouse warehouse(testLocation, initialCapacity);
00227     warehouse.addProduct(productName, productPrice, productQuantity, productId);
00228     EXPECT_EQ(warehouse.getPrice(productId), productPrice);
00229 }
00230
```

```
00234 TEST(WarehouseTest, getQuantityShouldReturnCorrectValue)
00235 {
00236     Warehouse warehouse(testLocation, initialCapacity);
00237     warehouse.addProduct(productName, productPrice, productQuantity, productId);
00238     EXPECT_EQ(warehouse.getQuantity(productId), productQuantity);
00239 }
```

8.56 test/WarehouseReport/WarehouseReportTest.cpp File Reference

Source file of tests for the WarehouseReport class.

```
#include <gtest/gtest.h>
#include "WarehouseReport/WarehouseReport.h"
#include <QDateTime>
```

Functions

- TEST (WarehouseReportTest, itLives)

  Test case to ensure that a WarehouseReport object can be instantiated.
- TEST (WarehouseReportTest, generateReportShouldReturnCorrectValue)

  Test case to verify that the WarehouseReport::generateReport method returns the correct CSV format.

8.56.1 Detailed Description

Source file of tests for the WarehouseReport class.

Definition in file WarehouseReportTest.cpp.

8.56.2 Function Documentation

8.56.2.1 TEST() [1/2]

```
TEST (
            WarehouseReportTest ,
            generateReportShouldReturnCorrectValue  )
```

Test case to verify that the WarehouseReport::generateReport method returns the correct CSV format.

Definition at line 27 of file WarehouseReportTest.cpp.

8.56.2.2 TEST() [2/2]

```
TEST (
            WarehouseReportTest ,
            itLives  )
```

Test case to ensure that a WarehouseReport object can be instantiated.

Definition at line 13 of file WarehouseReportTest.cpp.

8.57 WarehouseReportTest.cpp

Go to the documentation of this file.

```
00001
00006 #include <gtest/gtest.h>
00007 #include "WarehouseReport/WarehouseReport.h"
00008 #include <QDateTime>
00009
00013 TEST(WarehouseReportTest, itLives)
00014 {
00015     int warehouseId = 1;
00016     double capacity = 1000.0;
00017     QList<Report::ProductReport> productList;
00018     double operationalCosts = 2000.0;
00019     double netProfit = 5000.0;
00020
00021     WarehouseReport report(warehouseId, capacity, productList, operationalCosts, netProfit);
00022 }
00023
00027 TEST(WarehouseReportTest, generateReportShouldReturnCorrectValue)
00028 {
00029     int warehouseId = 1;
00030     double capacity = 1000.0;
00031     QList<Report::ProductReport> productList =
00032         {
00033             {"Product1", 10.0, 5},
00034             {"Product2", 20.0, 3}
00035         };
00036
00037     double operationalCosts = 2000.0;
00038     double netProfit = 5000.0;
00039
00040     WarehouseReport report(warehouseId, capacity, productList, operationalCosts, netProfit);
00041     QString generatedCSV = report.generateReport();
00042
00043     QString expectedCSVStart = QString("Warehouse ID,Capacity\n%1,%2\nProduct
      Name,Price,Quantity\n").arg(warehouseId).arg(capacity);
00044     ASSERT_TRUE(generatedCSV.startsWith(expectedCSVStart));
00045 }
00046
```

Index