Choose Application Theme

Order Processing System

Rationale for your choice

Stack Operations (Push, Pop, Peek)

Push: The **stack** data structure is ideal for tracking **recently completed orders**, as the last order to be completed should be the first one we can review or act upon (this follows the **Last-In-First-Out (LIFO)** principle). The **Push** operation allows us to "push" completed orders onto the stack, where they are temporarily stored.

> **Pop**: When an order is completed and needs to be removed from the tracking system (either for archival purposes or to "undo" the operation), the **Pop** operation is used. It ensures that the most recently completed order is removed first, preserving the LIFO order.
>
> **Peek**: The **Peek** operation is useful for quickly checking the most recently completed order without actually removing it from the stack. This allows users or administrators to review the most recent transactions without losing data. In the case of an order processing system, this is useful when managers or staff want to quickly see the last processed order before making decisions or reviewing data.

Queue Operations (Enqueue, Dequeue, Peek)

> **Enqueue**: The **queue** data structure naturally aligns with the way orders are received in a real-world order processing system. Orders must be processed in the order they are received, which is the **First-In-First-Out (FIFO)** principle. The **Enqueue** operation allows new orders to be added to the back of the queue, maintaining the sequence of incoming orders.
>
> **Dequeue**: The **Dequeue** operation removes the first order from the queue, representing the next order to be processed. This operation ensures that the orders are processed in the same order in which they were received (FIFO). By using **Dequeue**, the system processes orders one by one, in the exact sequence they arrive.
>
> **Peek**: The **Peek** operation allows the user to view the first order in the queue without removing it. This feature is useful for users who want to check the next order to be processed before making decisions or adjustments.

---

2. Interactivity and User Experience with GUI

Buttons for Stack and Queue Operations

In order to make the Order Processing System intuitive and interactive, we propose a **Graphical User Interface (GUI)** that provides buttons for each of the core operations:

- **Push Button**: When an order is completed, users can click the **Push** button to add the completed order to the stack.
- **Pop Button**: Users can click the **Pop** button to remove the most recently completed order from the stack if needed, allowing for quick access to historical data or undoing an order processing.
- **Enqueue Button**: This button will allow the user to add new orders to the system, ensuring the orders are processed in the order they are received.
- **Dequeue Button**: This button will remove the first order from the queue for processing, ensuring that the first order added is the first one to be handled.

This interaction model ensures that the user can easily control the flow of orders and completed orders, making the application both functional and user-friendly.

Displaying Stack and Queue States in the GUI

A key feature of the GUI will be to **display the current state of the stack and the queue**, providing real-time feedback to the user. This allows the user to:

- See the **current queue of incoming orders**, so they know which orders are waiting to be processed.
- View the **stack of recently completed orders**, so they can quickly reference the most recent transactions.

---

3. Flexibility for Future Enhancements

The system is designed with scalability in mind. By using stack and queue operations, it becomes easier to extend the functionality of the Order Processing System. Future enhancements could include:

- **Priority Queues**: Orders could be prioritized using a custom ordering system, with high-priority orders being dequeued before others.
- **Undo/Redo System**: The stack can be used to implement an "undo" system where users can reverse their last actions, such as canceling a completed order and re-queuing it.
- **Order History and Archiving**: The stack could be extended to hold completed orders for archival purposes, with options to store them in a database for long-term tracking and analytics.

---

4. Conclusion: Why This System Was Chosen

The **Order Processing System** was chosen because it is a simple yet effective way to showcase the power and utility of both stack and queue operations in a real-world context. The system benefits from the efficiency of queue operations for sequential order processing and the flexibility of stack operations for managing completed orders. Additionally, the graphical interface provides a straightforward way for users to interact with these operations, ensuring ease of use and a positive user experience

Explanation of the Implemented Application and its feature

In this application, we implemented an **Order Processing System** that utilizes both **stack** and **queue** operations to manage customer orders. The system is designed to process incoming orders in the sequence they are received and track completed orders for review or reprocessing. Additionally, a **Graphical User Interface (GUI)** was created to allow users to interact with the application easily, providing a clear, visual representation of both the order queue and completed orders stack

Three test cases for the chosen application

Three Test Cases for the Application
Test Case 1: Verify that orders are added to the queue in the correct sequence.
Test Case 2: Confirm that orders are processed in the correct order and moved to the stack.
Test Case 3: Check that the current state of the queue and stack displays accurately after multiple operations.

Challenges Faced during development

One of the key challenges we faced during development was slow internet connectivity. The slow speeds made it difficult to maintain real-time communication, whether through video calls, chat, or even file sharing. This resulted in delayed responses, fragmented discussions, and slower decision-making, which impacted the overall pace of development

Roles of Each Member and their contributions

Prince Jordan Japor - Developer
John Paulo Manalo - Designer
Gregg Jose - Tester/Theme

Testing Stack Operations

a. pushOrder() - Add to Stack
Test Case 1: Valid input

Scenario: Push a valid order (non-empty string) onto the stack.
Expected Outcome: The stack should increase in size by 1. The latest order in the stack should be the one just pushed.
Test Case 2: Invalid input (empty string)

Scenario: Attempt to push an empty string or null value to the stack.
Expected Outcome: The stack size should not increase. The system should either ignore the input or display an error message (this depends on your application's error-handling design).
Test Case 3: Stack overflow (if applicable)

Scenario: If there's a maximum size for the stack, try pushing orders until it reaches that size.
Expected Outcome: After the stack is full, pushing additional orders should either be ignored or trigger an error message.
b. popOrder() - Remove from Stack
Test Case 1: Valid pop

Scenario: Pop an order from a non-empty stack.
Expected Outcome: The stack size should decrease by 1, and the most recently added order should be removed.
Test Case 2: Pop from empty stack

Scenario: Attempt to pop an order from an empty stack.
Expected Outcome: The system should not crash. It could display an error message such as "Stack is empty" or simply return without doing anything.
Test Case 3: Stack after multiple pops

Scenario: Push several orders, then pop all orders one by one.
Expected Outcome: After popping all orders, the stack should be empty.
c. viewLatestOrder() - View the latest order in the stack
Test Case 1: Valid view

Scenario: After pushing several orders, view the latest order.
Expected Outcome: The latest order should be correctly displayed.
Test Case 2: View on an empty stack

Scenario: Attempt to view the latest order when the stack is empty.
Expected Outcome: The system should either return a default message (e.g., "No orders in stack") or handle the empty case gracefull

1. Test pushOrder() - Stack Push Operation
Test Case 1: Valid Input (Push)

Description: Push a non-empty order to the stack.
Steps:
Enter a valid string (e.g., "Order 1") in the "Complete Order" input field.
Click the "Push Order" button.
Expected Result:
The order should be added to the stack.
The UI should update to show the new stack state.
The input field should be cleared.
Test Case 2: Invalid Input (Empty String)

Description: Try to push an empty string or a blank order.
Steps:
Leave the "Complete Order" input field empty.
Click the "Push Order" button.
Expected Result:
A warning message should pop up with the message: "Order cannot be empty!".
The stack should remain unchanged.
The input field should remain empty.
2. Test popOrder() - Stack Pop Operation
Test Case 1: Pop from Non-Empty Stack

Description: Pop an order from a stack that contains one or more orders.
Steps:
Push a few orders (e.g., "Order 1", "Order 2").
Click the "Pop Order" button.
Expected Result:
The most recent order should be removed from the stack.
The UI should update to show the new stack state, reflecting the remaining orders.
The stack size should decrease by 1.
Test Case 2: Pop from Empty Stack

Description: Attempt to pop from an empty stack.
Steps:
Ensure the stack is empty.
Click the "Pop Order" button.

Expected Result:
A warning message should pop up: "Order Stack is Empty!".
The stack should remain unchanged.
3. Test viewLatestOrder() - View Latest Stack Order
Test Case 1: View Latest Order from Non-Empty Stack

Description: View the latest order in a non-empty stack.
Steps:
Push a few orders into the stack (e.g., "Order 1", "Order 2").
Click the "View Latest Order" button.
Expected Result:
The latest order should be displayed in the UI (e.g., "Latest Order: Order 2").
Test Case 2: View Latest Order from Empty Stack

Description: Attempt to view the latest order from an empty stack.
Steps:
Ensure the stack is empty.
Click the "View Latest Order" button.
Expected Result:
The UI should display: "No Recent Orders".
4. Test enqueueOrder() - Queue Enqueue Operation
Test Case 1: Valid Input (Enqueue)

Description: Enqueue a non-empty order to the queue.
Steps:
Enter a valid string (e.g., "Order A") in the "Pending Order" input field.
Click the "Enqueue Order" button.
Expected Result:
The order should be added to the queue.
The UI should update to reflect the new queue state.
The input field should be cleared.
Test Case 2: Invalid Input (Empty String)

Description: Try to enqueue an empty string or a blank order.
Steps:
Leave the "Pending Order" input field empty.
Click the "Enqueue Order" button.
Expected Result:
A warning message should appear: "Order cannot be empty!".
The queue should remain unchanged.
The input field should stay empty.
5. Test dequeueOrder() - Queue Dequeue Operation
Test Case 1: Dequeue from Non-Empty Queue

Description: Dequeue an order from a queue that contains one or more orders.
Steps:
Enqueue a few orders (e.g., "Order A", "Order B").
Click the "Dequeue Order" button.
Expected Result:
The first order in the queue should be removed.
The UI should update to show the new queue state, reflecting the remaining orders.
The queue size should decrease by 1.
Test Case 2: Dequeue from Empty Queue

Description: Attempt to dequeue from an empty queue.
Steps:
Ensure the queue is empty.

Click the "Dequeue Order" button.
Expected Result:
A warning message should appear: "Order Queue is Empty!".
The queue should remain unchanged.
6. Test viewNextOrder() - View Next Queue Order
Test Case 1: View Next Order from Non-Empty Queue

Description: View the next order in a non-empty queue.
Steps:
Enqueue a few orders (e.g., "Order A", "Order B").
Click the "View Next Order" button.
Expected Result:
The next order in the queue should be displayed (e.g., "Next Order: Order A").
Test Case 2: View Next Order from Empty Queue

Description: Attempt to view the next order from an empty queue.
Steps:
Ensure the queue is empty.
Click the "View Next Order" button.
Expected Result:
The UI should display: "No Pending Orders".
7. Edge Case Tests
Test Case 1: Push/Enqueue an Empty String

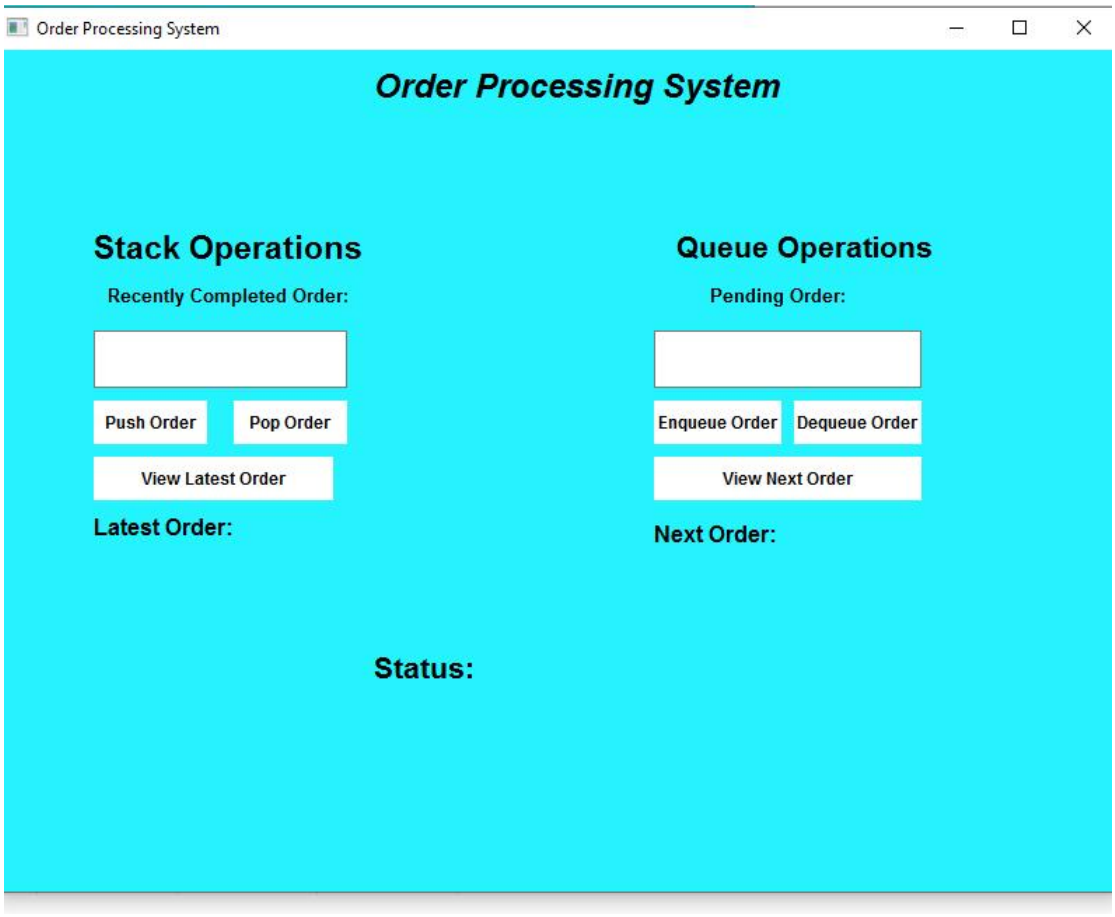
Description: Test the behavior of the push or enqueue operation when an empty string is provided as input.
Steps:
Leave the "Complete Order" or "Pending Order" input field empty and attempt to push or enqueue.
Expected Result:
A warning message should appear: "Order cannot be empty!".
The stack or queue should remain unchanged.
Test Case 2: Pop/Dequeue from Empty Stack/Queue

Description: Test the behavior when attempting to pop from an empty stack or dequeue from an empty queue.
Steps:
Ensure the stack or queue is empty.
Attempt to pop or dequeue.
Expected Result:
A warning message should appear: "Order Stack is Empty!" or "Order Queue is Empty!".
No changes should occur to the stack or queue.
Test Case 3: View Latest/Next Order from Empty Stack/Queue

Description: Attempt to view the latest order in an empty stack or the next order in an empty queue.
Steps:
Ensure the stack or queue is empty.
Click "View Latest Order" or "View Next Order".
Expected Result:
The UI should display: "No Recent Orders" for the stack or "No Pending Orders" for the queue.
Test Case 4: UI Updates After Operations

Description: Ensure that the UI updates correctly after each operation.
Steps:
Perform various stack and queue operations (push, pop, enqueue, dequeue).
Observe if the stack and queue displays in the UI are updated correctly after each operation.
Expected Result:
The stack and queue should be displayed accurately in the Status_2 label after each operation.

The latest and next order should be updated correctly in the corresponding labels.
Bug Reporting and Documentation
Bug: If the UI does not update correctly after performing stack/queue operations (e.g., the Status_2 label does not reflect the current state after pushing or enqueueing), this should be reported to the developer.
Bug: If the "View Latest Order" or "View Next Order" does not reflect the correct order after pushing/enqueueing, this would also need investigation.
Bug: If invalid input is not properly handled (e.g., pushing an empty string or leaving input fields blank), the input validation should be reviewed and fixed.



mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QMessageBox>
#include <QStack>
#include <QQueue>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    connect(ui->pushorder_2, &QPushButton::clicked, this, &MainWindow::pushOrder);
    connect(ui->poporder_2, &QPushButton::clicked, this, &MainWindow::popOrder);
    connect(ui->enqueueorder_2, &QPushButton::clicked, this, &MainWindow::enqueueOrder);
    connect(ui->dequeueorder_2, &QPushButton::clicked, this, &MainWindow::dequeueOrder);
    connect(ui->viewlatest_2, &QPushButton::clicked, this, &MainWindow::viewLatestOrder);
    connect(ui->viewnextorder_2, &QPushButton::clicked, this, &MainWindow::viewNextOrder);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::pushOrder()
{
    QString order = ui->forcompleteorder_2->text();
    if (order.isEmpty()) {
        QMessageBox::warning(this, "Warning", "Order cannot be empty!");
        return;
    }
    orderStack.push(order);
    updateStackDisplay();
    ui->forcompleteorder_2->clear();
}

void MainWindow::popOrder()
{
    if (orderStack.isEmpty()) {
        QMessageBox::warning(this, "Warning", "Order Stack is Empty!");
        return;
    }
    orderStack.pop();
    updateStackDisplay();
}

void MainWindow::enqueueOrder()
{
    QString order = ui->forpending_2->text();
    if (order.isEmpty()) {
        QMessageBox::warning(this, "Warning", "Order cannot be empty!");
        return;
    }
    orderQueue.enqueue(order);
    updateQueueDisplay();
    ui->forpending_2->clear();
```

```
}

void MainWindow::dequeueOrder()
{
    if (orderQueue.isEmpty()) {
        QMessageBox::warning(this, "Warning", "Order Queue is Empty!");
        return;
    }
    orderQueue.dequeue();
    updateQueueDisplay();
}

void MainWindow::viewLatestOrder()
{
    if (orderStack.isEmpty()) {
        ui->LatestOrder_2->setText("No Recent Orders");
    } else {
        ui->LatestOrder_2->setText("Latest Order: " + orderStack.top());
    }
}

void MainWindow::viewNextOrder()
{
    if (orderQueue.isEmpty()) {
        ui->Next_2->setText("No Pending Orders");
    } else {
        ui->Next_2->setText("Next Order: " + orderQueue.head());
    }
}

void MainWindow::updateStackDisplay()
{
    QString stackDisplay = "Order Stack: ";
    if (orderStack.isEmpty()) {
        stackDisplay += "Empty";
    } else {
        QStack<QString> tempStack = orderStack;
        while (!tempStack.isEmpty()) {
            stackDisplay += tempStack.pop() + " -> ";
        }
        stackDisplay = stackDisplay.left(stackDisplay.length() - 4); // Remove trailing " -> "
    }
    ui->Status_2->setText(stackDisplay);  // Display stack status in Status_2 label
}

void MainWindow::updateQueueDisplay()
{
    QString queueDisplay = "Order Queue: ";
    if (orderQueue.isEmpty()) {
        queueDisplay += "Empty";
    } else {
        QQueue<QString> tempQueue = orderQueue;
        while (!tempQueue.isEmpty()) {
            queueDisplay += tempQueue.dequeue() + " -> ";
        }
        queueDisplay = queueDisplay.left(queueDisplay.length() - 4); // Remove trailing " -> "
    }
    ui->Status_2->setText(queueDisplay);  // Display queue status in Status_2 label
}
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QStack>
#include <QQueue>
#include <QString>
#include <QMessageBox>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    // Stack operations
    void pushOrder();
    void popOrder();

    // Queue operations
    void enqueueOrder();
    void dequeueOrder();

    // View latest order (Stack)
    void viewLatestOrder();

    // View next order (Queue)
    void viewNextOrder();

private:
    Ui::MainWindow *ui;

    // Stack and Queue to hold orders
    QStack<QString> orderStack;
    QQueue<QString> orderQueue;

    // Helper functions to update the displays
    void updateStackDisplay();
    void updateQueueDisplay();
};

#endif
```

mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>800</width>
    <height>600</height>
   </rect>
  </property>
  <property name="font">
   <font>
    <family>a Another Tag</family>
    <bold>false</bold>
   </font>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QWidget" name="centralwidget_2" native="true">
    <property name="geometry">
     <rect>
      <x>0</x>
      <y>60</y>
      <width>791</width>
      <height>551</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <family>Academy Engraved LET</family>
      <bold>false</bold>
     </font>
    </property>
    <property name="styleSheet">
     <string notr="true">
background-color: rgb(15, 73, 143);
background-color: rgb(36, 92, 158);</string>
    </property>
    <widget class="QPushButton" name="viewlatest_2">
     <property name="geometry">
      <rect>
       <x>70</x>
       <y>290</y>
       <width>171</width>
       <height>31</height>
      </rect>
     </property>
     <property name="styleSheet">
      <string notr="true">QPushButton
{
            border:none;
            color:black;
            background-color:white;
            font: 700 9pt &amp;quot;Arial&amp;quot;;
```

```
            border-radius:40px;
}
QPushButton:pressed
{
            background-color:silver;
}</string>
     </property>
     <property name="text">
      <string>View Latest Order</string>
     </property>
    </widget>
    <widget class="QLabel" name="Title_2">
     <property name="geometry">
      <rect>
       <x>270</x>
       <y>0</y>
       <width>291</width>
       <height>51</height>
      </rect>
     </property>
     <property name="font">
      <font>
       <family>Arial</family>
       <pointsize>18</pointsize>
       <italic>true</italic>
       <bold>true</bold>
      </font>
     </property>
     <property name="text">
      <string>Order Processing System</string>
     </property>
    </widget>
    <widget class="QLabel" name="Stack_2">
     <property name="geometry">
      <rect>
       <x>70</x>
       <y>120</y>
       <width>201</width>
       <height>41</height>
      </rect>
     </property>
     <property name="font">
      <font>
       <family>Arial</family>
       <pointsize>17</pointsize>
       <bold>true</bold>
      </font>
     </property>
     <property name="text">
      <string>Stack Operations</string>
     </property>
    </widget>
    <widget class="QLineEdit" name="forcompleteorder_2">
     <property name="geometry">
      <rect>
       <x>70</x>
       <y>200</y>
       <width>181</width>
       <height>41</height>
      </rect>
```

```
    </property>
    <property name="font">
     <font>
      <family>a Another Tag</family>
      <bold>false</bold>
     </font>
    </property>
    <property name="styleSheet">
     <string notr="true">background-color: rgb(255, 255, 255);</string>
    </property>
   </widget>
   <widget class="QLabel" name="RecentOrder_2">
    <property name="geometry">
     <rect>
      <x>80</x>
      <y>160</y>
      <width>181</width>
      <height>31</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <family>Arial</family>
      <pointsize>10</pointsize>
      <bold>true</bold>
     </font>
    </property>
    <property name="text">
     <string>Recently Completed Order:</string>
    </property>
   </widget>
   <widget class="QPushButton" name="pushorder_2">
    <property name="geometry">
     <rect>
      <x>70</x>
      <y>250</y>
      <width>81</width>
      <height>31</height>
     </rect>
    </property>
    <property name="autoFillBackground">
     <bool>false</bool>
    </property>
    <property name="styleSheet">
     <string notr="true">QPushButton
{
        border:black;
        color:black;
        background-color:white;
        font: 700 9pt &amp;quot;Arial&amp;quot;;
        border-radius:40px;
}
QPushButton:pressed
{
        background-color:silver;
}</string>
    </property>
    <property name="text">
     <string>Push Order</string>
    </property>
```

```
   </widget>
   <widget class="QPushButton" name="poporder_2">
    <property name="geometry">
     <rect>
      <x>170</x>
      <y>250</y>
      <width>81</width>
      <height>31</height>
     </rect>
    </property>
    <property name="styleSheet">
     <string notr="true">QPushButton
{
	border:none;
	color:black;
	background-color:white;
	font: 700 9pt &amp;quot;Arial&amp;quot;;
	border-radius:40px;
}
QPushButton:pressed
{
	background-color:silver;
}</string>
    </property>
    <property name="text">
     <string>Pop Order</string>
    </property>
   </widget>
   <widget class="QLabel" name="LatestOrder_2">
    <property name="geometry">
     <rect>
      <x>70</x>
      <y>330</y>
      <width>161</width>
      <height>21</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <family>Arial</family>
      <pointsize>12</pointsize>
      <bold>true</bold>
     </font>
    </property>
    <property name="text">
     <string>Latest Order:</string>
    </property>
   </widget>
   <widget class="QLabel" name="Queue_2">
    <property name="geometry">
     <rect>
      <x>480</x>
      <y>120</y>
      <width>191</width>
      <height>41</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <family>Arial</family>
```

```
   <pointsize>16</pointsize>
   <bold>true</bold>
  </font>
 </property>
 <property name="text">
  <string> Queue Operations</string>
 </property>
</widget>
<widget class="QLabel" name="Pending_2">
 <property name="geometry">
  <rect>
   <x>510</x>
   <y>160</y>
   <width>111</width>
   <height>31</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <family>Arial</family>
   <pointsize>10</pointsize>
   <bold>true</bold>
  </font>
 </property>
 <property name="text">
  <string>Pending Order:</string>
 </property>
</widget>
<widget class="QLineEdit" name="forpending_2">
 <property name="geometry">
  <rect>
   <x>470</x>
   <y>200</y>
   <width>191</width>
   <height>41</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <family>Academy Engraved LET</family>
   <bold>false</bold>
  </font>
 </property>
 <property name="styleSheet">
  <string notr="true">background-color: rgb(255, 255, 255);</string>
 </property>
 <property name="text">
  <string/>
 </property>
</widget>
<widget class="QPushButton" name="dequeueorder_2">
 <property name="geometry">
  <rect>
   <x>570</x>
   <y>250</y>
   <width>91</width>
   <height>31</height>
  </rect>
 </property>
 <property name="styleSheet">
```

```
     <string notr="true">QPushButton
{
            border:none;
            color:black;
            background-color:white;
            font: 700 9pt &amp;quot;Arial&amp;quot;;
            border-radius:40px;
}
QPushButton:pressed
{
            background-color:silver;
}</string>
    </property>
    <property name="text">
     <string>Dequeue Order</string>
    </property>
   </widget>
   <widget class="QPushButton" name="enqueueorder_2">
    <property name="geometry">
     <rect>
      <x>470</x>
      <y>250</y>
      <width>91</width>
      <height>31</height>
     </rect>
    </property>
    <property name="styleSheet">
     <string notr="true">QPushButton
{
            border:none;
            color:black;
            background-color:white;
            font: 700 9pt &amp;quot;Arial&amp;quot;;
            border-radius:40px;
}
QPushButton:pressed
{
            background-color:silver;
}</string>
    </property>
    <property name="text">
     <string>Enqueue Order</string>
    </property>
   </widget>
   <widget class="QPushButton" name="viewnextorder_2">
    <property name="geometry">
     <rect>
      <x>470</x>
      <y>290</y>
      <width>191</width>
      <height>31</height>
     </rect>
    </property>
    <property name="styleSheet">
     <string notr="true">QPushButton
{
            border:none;
            color:black;
            background-color:white;
            font: 700 9pt &amp;quot;Arial&amp;quot;;
```

```
            border-radius:40px;
}
QPushButton:pressed
{
            background-color:silver;
}</string>
     </property>
     <property name="text">
      <string>View Next Order</string>
     </property>
    </widget>
    <widget class="QLabel" name="Next_2">
     <property name="geometry">
      <rect>
       <x>470</x>
       <y>330</y>
       <width>191</width>
       <height>31</height>
      </rect>
     </property>
     <property name="font">
      <font>
       <family>Arial</family>
       <pointsize>12</pointsize>
       <bold>true</bold>
      </font>
     </property>
     <property name="text">
      <string>Next Order:</string>
     </property>
    </widget>
    <widget class="QLabel" name="Status_2">
     <property name="geometry">
      <rect>
       <x>270</x>
       <y>420</y>
       <width>221</width>
       <height>41</height>
      </rect>
     </property>
     <property name="font">
      <font>
       <family>Arial</family>
       <pointsize>16</pointsize>
       <bold>true</bold>
      </font>
     </property>
     <property name="text">
      <string>Status:</string>
     </property>
    </widget>
   </widget>
   <widget class="QMenuBar" name="menubar_2">
    <property name="geometry">
     <rect>
      <x>20</x>
      <y>30</y>
      <width>800</width>
      <height>22</height>
     </rect>
```

```
   </property>
  </widget>
  <widget class="QStatusBar" name="statusbar_2">
   <property name="geometry">
    <rect>
     <x>20</x>
     <y>30</y>
     <width>3</width>
     <height>27</height>
    </rect>
   </property>
  </widget>
 </widget>
 <widget class="QStatusBar" name="statusbar"/>
</widget>
<resources/>
<connections/>
</ui>
```