ACKNOWLEDGEMENT

No Art is ever complete without the Master artist's touches. And to the Master of Masters **BHAGAWAN SRI SATHYA SAI BABA**, I offer my deep sense of gratitude. The wonderful faculty in the school that are handpicked by HIM have been or pillars of support throughout this journey.

I would love to utilize this opportunity to offer my gratitude to the principal, **SHRI SIVARAMAKRISHNAIAH** for presenting me this wonderful opportunity which was filled with fun and learning at the same time.

A special vote of thanks is due to my Informatics teacher, **SHRI SAI PAVAN** for guiding me all along this wonderful saga of exploration and deep dive into this subject.

No journey can be smooth throughout and during the rough phases one needs to shoulder to rely on. And to those shoulders of my **family** and my **classmates**, I offer my deepest of thanks.

Table Of Contents

Aim

To create a basic Auction Software using socket programming from Python.

Introduction

An auction that can be held across different computers in a same network is something that can never be dreamt of because no one would have got that idea. This software is designed and coded for that purpose of bidding in an auction right in your homes in your computer. With just a click you can bid for anything , you can be someone who is auctioning something or you can be taking part in some auction started by someone. You can add a new item you want to auction along with the description to it .Everything has an end so is the item which is getting auctioned if its not auctioned in a particular time then you will lose the opportunity of acquiring it.

WHY is it necessary?

It provides the necessary tools for sellers to list items, buyers to bid on them, and administrators to monitor them. The software **helps automate the bidding process**, allowing buyers to bid from any location at a given time without having to be physically present at the auction

How does it work?

Firstly if you want to auction a product you need to add that item to the products using an ADD ITEM feature available in the software, Next you need to go to the auction page in the software search up the item you would like to auction and click on the start auction button in that page and the starts the server and waits for clients to join after the waiting is done once the minimum number of clients is reached then the auction starts and then you can happily enjoy bidding.

Scope of the project

This software is designed to reduce the pressure of people taking part in an auction to go to a particular place then bid for the item they can do it at ease sitting in their own houses. Our software enables you to have an auction amongst any number of people at once. It is efficient enough to give everyone a smooth experience you can save a lot of time and money using this software.

Softwares & Libraries

❖ PyCharm Community Edition 2023.3.3

PyCharm is an integrated development surroundings (IDE) used for programming in Python. It is advanced by means of the Czech organization JetBrains and is to be had for Windows, mac OS, and Linux. PyCharm is to be had in two variations: Community Edition and Professional Edition. The Community Edition is loose and open supply, while the Professional Edition is a paid subscription carrier.

❖ Python 3.12

Python is a set of instructions that we give in the form of a Program to our computer to perform any specific task. It is a Programming language having properties like it is interpreted, object-oriented and it is high-level too. Due to its beginner-friendly syntax, it became a clear choice for beginners to start their programming journey. The major focus behind creating it is making it easier for developers to read and understand, also reducing the lines of code.

❖ PyQt5

PyQt5 is a set of Python bindings for the Qt application framework, allowing Python to be used for cross-platform desktop and mobile application development. It provides tools to create user interfaces that can run on various operating systems, including Windows, mac OS, Linux, iOS, and Android.

❖ Sys

The sys module in Python provides various functions and variables that are used to manipulate different parts of the Python runtime environment. It allows operating on the interpreter as it provides access to the variables and functions that interact strongly with the interpreter.

❖ QtDesigner

Qt Designer is a tool for designing and building graphical user interfaces (GUIs) with Qt Widgets. It provides a **what-you-see-is-what-you-get (WYSIWYG) interface** to create GUIs for PyQt applications by dragging and dropping QWidget objects on an empty form.

❖ MySQL

MySQL is a popular open-source Relational Database Management System (RDBMS) that uses **SQL** (Structured Query Language) for database operations. While MySQL is a specific database system accessible for free and supports various programming languages.

❖ Sockets

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket (node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server. They are the real backbones behind web browsing. In simpler terms, there is a server and a client.

❖ Threading

A thread is an entity within a process that can be scheduled for execution. Also, it is the smallest unit of processing that can be performed in an OS (Operating System). In simple words, a thread is a sequence of such instructions within a program that can be executed independently of other code.

❖ Traceback

Traceback is a python module that provides a standard interface to extract, format and print stack traces of a python program. When it prints the stack trace it exactly mimics the behaviour of a python interpreter. Useful when you want to print the stack trace at any step. They are usually seen when an exception occurs.

❖ Subprocess

The subprocess module present in Python is used to run new applications or programs through Python code by creating new processes. It also helps to obtain the input/output/error pipes as well as the exit codes of various commands.

Program design

❖ Login Page

The Login Page is the entry point for users, where they enter their credentials—username and password. The system checks these against stored data in a secure database. Successful authentication grants access to the software, ensuring that only authorized individuals can enter and use the platform.

❖ Auction Page

The Auction Page allows users to start and manage auctions. Here, users can initiate an auction event and engage with all registered clients. This page typically features options to set the auction parameters, such as starting time, duration, and rules, facilitating seamless interaction and participation.

❖ Products Page

The Products Page showcases all items available for auction. It provides users with detailed information, including descriptions, images, and starting bid prices. This comprehensive view helps users make informed decisions when placing bids, as they can easily compare products and assess their value.

❖ Add Item Page

The Add Item Page enables users to submit new items for auction. Users can fill out necessary details, such as the item's name, description, starting bid, and images. This feature encourages participation by allowing users to contribute items, thereby diversifying the auction inventory and enhancing the auction experience for everyone.

❖ Bids Page

The Bids Page displays all items that users have successfully purchased in previous auctions. It includes information about each transaction, such as item names, final prices, and auction dates. This functionality helps users keep track of their purchases and manage their auction activities efficiently.

❖ Profile Page

The Profile Page allows users to manage their personal information and preferences. Users can edit their profiles, update contact details, and adjust settings related to their auction activities. Additionally, this page enables other users to view profile information, fostering a sense of community and trust within the auction environment.

Database

❖ Users

Field	Type	Null	Key	Default	Extra
user_id	varchar(10)	YES		NULL	
name	varchar(20)	YES		NULL	
gmail	varchar(30)	YES		NULL	
date_of_birth	date	YES		NULL	
password	varchar(16)	YES		NULL	
active	enum('yes','no')	YES		no	

❖ Products

Field	Type	Null	Key	Default	Extra
product_id	int(11)	NO	PRI	NULL	
product	varchar(50)	YES		NULL	
last_date	date	YES		NULL	
product_desc	varchar(500)	YES		NULL	
status	varchar(6)	YES		unsold	
base_price	varchar(10)	YES		NULL	

❖ Images

Field	Type	Null	Key	Default	Extra
image_id	int(11)	YES		NULL	
image	longblob	YES		NULL	

Program Code

```
# IMPORTING MODULES FOR THE PROJECT

import time

import traceback

from PyQt5 import QtWidgets, QtGui, QtCore

from PyQt5.QtWidgets import (QLabel, QPushButton, QLineEdit, QFileDialog,
QWidget, QScrollBar, QDialog, QMainWindow,

                             QMessageBox, QScrollArea, QApplication,
QCalendarWidget, QTextEdit,QVBoxLayout,QHBoxLayout,QCheckBox,QDateEdit)

from PyQt5.QtGui import QPixmap, QFont,QIcon,QImage,QPalette,QColor

from PyQt5.QtCore import QRect, Qt , QSize, QDate

import sys

import mysql.connector as sql

import socket

import threading

import pickle

import time

import os

import select


# CREATING DATABASE

try:

    connection = sql.connect(host="192.168.102.206", user="root",
password="sairam")

    main_cursor = connection.cursor()


    db_creating_query = "CREATE DATABASE IF NOT EXISTS aucsof_ip"
```

```
main_cursor.execute(db_creating_query)


selecting_db = "USE aucsof_ip"
main_cursor.execute(selecting_db)


creating_tbl_for_users = ('''
                    CREATE TABLE IF NOT EXISTS users (
                        id INT AUTO_INCREMENT PRIMARY KEY,
                        full_name VARCHAR(255),
                        email VARCHAR(255),
                        phone VARCHAR(20),
                        address VARCHAR(255),
                        dob DATE,
                        bio TEXT,
                        linkedin VARCHAR(255),
                        password VARCHAR(255),
                        active ENUM('yes','no') DEFAULT 'no'
                    )
                ''')

main_cursor.execute(creating_tbl_for_users)


creating_tbl_for_products = ("CREATE TABLE IF NOT EXISTS products("
                             "product_id VARCHAR(6) ,"
                             "product VARCHAR(50),"
                             "last_date DATE,"
                             "product_desc VARCHAR(500),"
                             "status VARCHAR(6) DEFAULT 'unsold',"
```

```
                                "base_price VARCHAR(10))")


    main_cursor.execute(creating_tbl_for_products)
    main_cursor.execute("alter table products modify column bid_increement
varchar(10) default '1000'")



    creating_tbl_for_prodimg = """
        CREATE TABLE IF NOT EXISTS IMAGES (
            image_id int,
            image LONGBLOB
        );
        """
    main_cursor.execute(creating_tbl_for_prodimg)




    connection.commit()


except sql.Error as e:
    print(f"Error: {e}")
```

```
#CREATING LOGIN UI
class login(QDialog):
    def __init__(self):
        super(login, self).__init__()
        self.setWindowTitle("Login") #LOGIN UI TITLE
        #self.setStyleSheet("background-color:rgb(234,241,255)")
        self.setGeometry(470,200,400,300)


        self.tabsbg_LBL = QLabel(self)
        self.tabsbg_LBL.setStyleSheet("background-
color:rgba(0,12,95,205)")
        self.tabsbg_LBL.setGeometry(0, 0, 400, 70)



        self.pointsize_headingLBL=18 #SETTING  FONT SIZE FOR HEADINGS
        self.pointsize_LBL=14 #SETTING  FONT SIZE FOR LABELS


        # CREATING FONTS FOR HEADING LABELS
        self.heading_LBLfont = QFont()
        self.heading_LBLfont.setFamily(u"Palatino Linotype")
        self.heading_LBLfont.setPointSize(22)
        self.heading_LBLfont.setBold(True)
        self.heading_LBLfont.setItalic(True)
        self.heading_LBLfont.setWeight(75)


        # CREATING FONTS FOR NORMAL LABELS
        self.LBL_font = QFont()
        self.LBL_font.setFamily(u"Palatino Linotype")
        self.LBL_font.setPointSize(14)
```

```
        self.LBL_font.setBold(True)
        self.LBL_font.setItalic(True)
        self.LBL_font.setWeight(75)

        # CREATING FONTS FOR LINE EDITS
        self.LEfont=QtGui.QFont()
        self.LEfont.setPointSize(12)
        self.LEfont.setFamily("Platino Linotype")



        self.Login_LBL=QLabel(self)
        self.Login_LBL.setText("LOGIN")
        self.Login_LBL.setFont(self.heading_LBLfont)
        self.Login_LBL.setGeometry(155,20,101,31)
        self.Login_LBL.setStyleSheet("background-color: rgba(255, 255,
255, 10);color : white")


        self.username_LBL=QLabel(self)
        self.username_LBL.setText("Username")
        self.username_LBL.setFont(self.LBL_font)
        self.username_LBL.setGeometry(50,110,90,23)


        self.pointsize_LBL-=6
        self.LBL_font.setItalic(False)
        self.username_LE=QtWidgets.QLineEdit(self)
        self.username_LE.setPlaceholderText("Enter your username")
        self.username_LE.setGeometry(150,110,120,25)
        self.username_LE.setFont(self.LEfont)
```

```
        self.password_LBL=QLabel(self)
        self.password_LBL.setText("Password")
        self.pointsize_LBL+=6
        self.LBL_font.setItalic(True)
        self.password_LBL.setGeometry(50,180,90,15)
        self.password_LBL.setFont(self.LBL_font)

        self.password_LE=QtWidgets.QLineEdit(self)
        self.password_LE.setPlaceholderText("Enter your password")
        self.LBL_font.setItalic(False)
        self.password_LE.setFont(self.LEfont)
        self.password_LE.setGeometry(150,175,120,25)
        self.password_LE.setEchoMode(QtWidgets.QLineEdit.Password)

        self.login_BTN = QPushButton(self)
        self.login_BTN.setText("Login")
        self.LBL_font.setItalic(True)
        self.login_BTN.setFont(self.LBL_font)
        self.login_BTN.setGeometry(130, 220, 80, 35)
        self.login_BTN.clicked.connect(self.mainpg)
        self.login_BTN.setStyleSheet("background-color: rgba(255, 255,
255, 10);padding 10px")

        self.sign_up_LBL=QLabel(self)
        self.sign_up_LBL.setText("Don't have an account Signup!")
        self.LBL_font.setItalic(True)
        self.sign_up_LBL.setFont(self.LBL_font)
        self.sign_up_LBL.setGeometry(30,270,280,30)
```

```
        self.pointsize_LBL-=5
        self.showpasswordcheckbox=QCheckBox(self)
        self.showpasswordcheckbox.setFont(self.LBL_font)
        self.showpasswordcheckbox.setText("Show")
        self.showpasswordcheckbox.setGeometry(275,175,150,25)
        self.showpasswordcheckbox.stateChanged.connect(self.showpwd)

        self.sign_up_BTN=QPushButton(self)
        self.sign_up_BTN.setText("Sign up")
        self.sign_up_BTN.setFont(self.LBL_font)
        self.sign_up_BTN.setGeometry(310, 265, 80, 35)
        self.sign_up_BTN.clicked.connect(self.signup_pg)
        self.sign_up_BTN.setStyleSheet("padding 30px")

    def signup_pg(self):
        try:
            self.destroy()
            self.open_signup_pg=signup_ui()
            self.open_signup_pg.show()
        except Exception as e:
            traceback.print_exc()

    def showpwd(self):
        try:
            if self.showpasswordcheckbox.isChecked():
                self.password_LE.setEchoMode(False)
            else:
                self.password_LE.setEchoMode(QtWidgets.QLineEdit.Password)
```

```
        except Exception as e:
            traceback.print_exc()


    #CHECKING CREDENTIALS
    def mainpg(self):
        try:
            query=f"select id, password from users where id=%s and
password=%s"

main_cursor.execute(query,(self.username_LE.text(),self.password_LE.text()
))
            res=main_cursor.fetchone()
            if res:
                self.destroy()
                self.open_mainpg = main_ui()
                self.open_mainpg.show()



            #else:
                #QtWidgets.QMessageBox.warning(
                    #self, 'Unsuccessful', 'Enter correct credentials!!')



        except Exception as e:
            traceback.print_exc()


    def userstodb(self):
        try:
            userinfo = self.username_LE.text()
```

```
            userpass = self.password_LE.text()
            update_tbl=f"update users set active='yes' where
user_id='{userinfo}'"
            #main_cursor.execute(update_tbl)
            #connection.commit()
        except Exception as e:
            traceback.print_exc()
```

```
#CREATING SIGN UP
class signup_ui(QDialog):
    def __init__(self):
        super(signup_ui, self).__init__()
        self.setWindowTitle("Sign up!")  # Sign up UI TITLE
        #self.setGeometry(470, 200, 500, 550)  # Increased height to
accommodate additional fields


        # Font size for headings, labels, and line edits
        self.pointsize_headingLBL = 18
        self.pointsize_LBL = 14
        self.pointsize_LE = 12


        self.tabsbg_LBL = QLabel(self)
        self.tabsbg_LBL.setStyleSheet("background-
color:rgba(0,12,95,205)")
        self.tabsbg_LBL.setGeometry(0, 0, 700, 70)


        # Setting up fonts
        self.heading_labels_font = QtGui.QFont()
        self.heading_labels_font.setPointSize(self.pointsize_headingLBL)
        self.heading_labels_font.setFamily("Platino Linotype")
        self.heading_labels_font.setBold(True)
        self.heading_labels_font.setItalic(True)


        self.LBL_font = QtGui.QFont()
        self.LBL_font.setPointSize(self.pointsize_LBL)
        self.LBL_font.setFamily("Platino Linotype")
        self.LBL_font.setItalic(True)
```

```
        self.LEfont = QtGui.QFont()
        self.LEfont.setPointSize(self.pointsize_LE)
        self.LEfont.setFamily("Platino Linotype")

        self.Signup_LBL = QLabel(self)
        self.Signup_LBL.setText("SIGN UP")
        self.Signup_LBL.setFont(self.heading_labels_font)
        self.Signup_LBL.setGeometry(200, 30, 120, 25)
        self.Signup_LBL.setStyleSheet("background-color: rgba(255, 255,
255, 10); color : white")

        # User ID, Name, and Email Fields
        self.userid_LBL = QLabel(self)
        self.userid_LBL.setText("Create user id")
        self.userid_LBL.setFont(self.LBL_font)
        self.userid_LBL.setGeometry(50, 120, 130, 15)
        self.userid_LE = QtWidgets.QLineEdit(self)
        self.userid_LE.setFont(self.LEfont)
        self.userid_LE.setPlaceholderText("Username")
        self.userid_LE.setGeometry(200, 118, 150, 25)

        self.name_LBL = QLabel(self)
        self.name_LBL.setText("Name")
        self.name_LBL.setFont(self.LBL_font)
        self.name_LBL.setGeometry(50, 165, 148, 15)
        self.name_LE = QtWidgets.QLineEdit(self)
        self.name_LE.setFont(self.LEfont)
        self.name_LE.setGeometry(200, 160, 150, 25)
```

```
self.name_LE.setPlaceholderText("First Name")


self.phone_LBL = QLabel(self)
self.phone_LBL.setText("Phone Number")
self.phone_LBL.setFont(self.LBL_font)
self.phone_LBL.setGeometry(50, 250, 148, 15)
self.phone_LE = QtWidgets.QLineEdit(self)
self.phone_LE.setFont(self.LEfont)
self.phone_LE.setGeometry(200, 245, 200, 25)
self.phone_LE.setPlaceholderText("Phone Number")


self.address_LBL = QLabel(self)
self.address_LBL.setText("Address")
self.address_LBL.setFont(self.LBL_font)
self.address_LBL.setGeometry(50, 290, 148, 15)
self.address_LE = QtWidgets.QLineEdit(self)
self.address_LE.setFont(self.LEfont)
self.address_LE.setGeometry(200, 285, 200, 25)
self.address_LE.setPlaceholderText("Address")


# Gmail, DOB, Password Fields
self.gmail_LBL = QLabel(self)
self.gmail_LBL.setText("Gmail")
self.gmail_LBL.setFont(self.LBL_font)
self.gmail_LBL.setGeometry(50, 410, 148, 15)
self.gmail_LE = QtWidgets.QLineEdit(self)
self.gmail_LE.setFont(self.LEfont)
self.gmail_LE.setGeometry(200, 405, 200, 25)
```

```
        self.gmail_LE.setPlaceholderText("Your Gmail")


        self.dob_LBL = QLabel(self)
        self.dob_LBL.setText("DOB")
        self.dob_LBL.setFont(self.LBL_font)
        self.dob_LBL.setGeometry(50, 450, 148, 15)
        self.duedate_LE = QDateEdit(QDate.currentDate(), self)
        self.duedate_LE.setCalendarPopup(True)
        self.duedate_LE.setDisplayFormat("yyyy-MM-dd")
        self.duedate_LE.setFont(self.LEfont)
        self.duedate_LE.setGeometry(200, 444, 200, 25)


        self.password_LBL = QLabel(self)
        self.password_LBL.setText("Password")
        self.password_LBL.setFont(self.LBL_font)
        self.password_LBL.setGeometry(50, 490, 148, 15)
        self.password_LE = QtWidgets.QLineEdit(self)
        self.password_LE.setFont(self.LEfont)
        self.password_LE.setGeometry(200, 485, 200, 25)
        self.password_LE.setPlaceholderText("Set password")
        self.password_LE.setEchoMode(QtWidgets.QLineEdit.Password)


        self.confirmpassword_LBL = QLabel(self)
        self.confirmpassword_LBL.setText("Confirm Password")
        self.confirmpassword_LBL.setFont(self.LBL_font)
        self.confirmpassword_LBL.setGeometry(40, 532, 153, 15)
        self.confirmpassword_LE = QtWidgets.QLineEdit(self)
        self.confirmpassword_LE.setFont(self.LEfont)
```

```
        self.confirmpassword_LE.setGeometry(200, 528, 200, 25)
        self.confirmpassword_LE.setPlaceholderText("Confirm password")
        self.confirmpassword_LE.setEchoMode(QtWidgets.QLineEdit.Password)


        self.signup_BTN = QPushButton(self)
        self.signup_BTN.setText("Sign up")
        self.signup_BTN.setFont(self.LBL_font)
        self.signup_BTN.clicked.connect(self.confirmation)
        self.signup_BTN.setGeometry(210, 580, 80, 40)
        self.signup_BTN.setStyleSheet("background-color: rgba(255, 255,
255, 10); padding 10px")


        self.back_BTN = QPushButton(self)
        self.back_BTN.setText("Back")
        self.back_BTN.setFont(self.LBL_font)
        self.back_BTN.setGeometry(700, 0, 100, 50)
        self.back_BTN.clicked.connect(self.going_back)
        self.back_BTN.setStyleSheet("background-color: rgba(255, 255, 255,
10); color: white")


    def going_back(self):
        try:
            self.close()
            self.loginobj = login()
            self.loginobj.show()
        except Exception as e:
            traceback.print_exc()


    def confirmation(self):
```

```
        try:
            if self.confirmpassword_LE.text() == "" or
self.password_LE.text() == "":
                QtWidgets.QMessageBox.information(self, 'Unsuccessful',
'Enter a password!')
                return

            if self.password_LE.text() == self.confirmpassword_LE.text():
                self.add_user()
                QtWidgets.QMessageBox.information(self, 'Successful',
'Signed up Successfully!')
        except Exception as e:
            traceback.print_exc()

    def add_user(self):
        try:
            username = self.userid_LE.text()
            first_name = self.name_LE.text()
            phone = self.phone_LE.text()
            address = self.address_LE.text()
            gmail = self.gmail_LE.text()
            dob = self.duedate_LE.text()
            password = self.password_LE.text()

            query = f"""
                INSERT INTO users (id, full_name, email , phone, address,
dob , password)
                VALUES ('{username}', '{first_name}','{gmail}',
'{phone}', '{address}', '{dob}', '{password}')
            """
```

```
            main_cursor.execute(query)
            connection.commit()
        except Exception as e:
            QtWidgets.QMessageBox.warning(self, 'Error', 'Something went
wrong!')
            print(f"Error: {e}")
```

```
#CREATING HOME PAGE UI
class main_ui(QMainWindow):
    def __init__(self):
        super(main_ui,self).__init__()
        self.setWindowTitle("Auction Software")
        self.setGeometry(275,140,800,500)

        self.setStyleSheet("background-color:rgb(234,241,255)")

        self.tabsbg_LBL = QLabel(self)
        self.tabsbg_LBL.setStyleSheet("background-
color:rgba(0,12,95,205)")
        self.tabsbg_LBL.setGeometry(0, 0, 791, 90)

        self.pointsize_LBL = 40  # SETTING  FONT SIZE FOR LABELS
        self.LBL_font = QtGui.QFont()
        self.LBL_font.setPointSize(self.pointsize_LBL)
        self.LBL_font.setFamily("Platino Linotype")
        self.LBL_font.setItalic(True)
        self.LBL_font.setBold(True)
        self.LBL_font.setWeight(65)

        self.label_font = QtGui.QFont()
        self.label_font.setPointSize(14)
        self.label_font.setFamily("Lucida Handwrting")

        self.home_btn=QPushButton(self)
        self.home_btn.setText("Home")
```

```
        self.home_btn.setStyleSheet("background-color: rgba(255, 255, 255,
10);color : white")
        self.home_btn.setGeometry(5,25,120,41)
        self.home_btn.setFont(self.label_font)
        #self.home_btn.setAttribute(Qt.WA_TranslucentBackground)


        self.auction_BTN = QPushButton(self)
        self.auction_BTN.setStyleSheet("background-color: rgba(255, 255,
255, 10);color : white")
        self.auction_BTN.setText("Auction")
        self.auction_BTN.setGeometry(129, 25, 120, 41)
        self.auction_BTN.setFont(self.label_font)
        self.auction_BTN.clicked.connect(self.connecto_auction)


        self.products_BTN = QPushButton(self)
        self.products_BTN.setStyleSheet("background-color: rgba(255, 255,
255, 10);color : white")
        self.products_BTN.setText("Products")
        self.products_BTN.setFont(self.label_font)
        self.products_BTN.setGeometry(253, 25, 120, 41)
        self.products_BTN.clicked.connect(self.connectto_products)


        self.add_item_BTN = QPushButton(self)
        self.add_item_BTN.setStyleSheet("background-color: rgba(255, 255,
255, 10);color : white")
        self.add_item_BTN.setText("Add Item")
        self.add_item_BTN.setFont(self.label_font)
        self.add_item_BTN.setGeometry(375, 25, 120, 41)
        self.add_item_BTN.clicked.connect(self.connec_to_additem)
```

```
        self.wallet_BTN = QPushButton(self)
        self.wallet_BTN.setStyleSheet("background-color: rgba(255, 255,
255, 10);color : white")
        self.wallet_BTN.setText("Wallet")
        self.wallet_BTN.setFont(self.label_font)
        self.wallet_BTN.setGeometry(500,25,120,41)
        self.wallet_BTN.clicked.connect(self.connecto_wallet)



        self.profile_BTN = QPushButton(self)
        self.profile_BTN.setStyleSheet("background-color: rgba(255, 255,
255, 10);color : white")
        self.profile_BTN.setText("Profile")
        self.profile_BTN.setFont(self.label_font)
        self.profile_BTN.setGeometry(625, 25, 120, 41)
        self.profile_BTN.clicked.connect(self.connecto_profile)


        self.exit_btn=QPushButton(self)
        self.exit_btn.setIcon(QIcon("C:/Users/XII
Info/Desktop/Project!/Pictures/exitpg2.png"))
        self.exit_btn.setStyleSheet("background-color: rgba(255, 255, 255,
10);color : white")
        self.exit_btn.setGeometry(750,25,40,40)
        self.exit_btn.clicked.connect(self.backtologin)



        self.auction_software_LBL=QLabel(self)
        self.auction_software_LBL.setFont(self.LBL_font)
        self.auction_software_LBL.setText("Auction Software")
        self.auction_software_LBL.setGeometry(50,200,500,100)
```

```
        self.hslogo_LBL=QLabel(self)
        self.hslogo_pic=QPixmap("finallogo_using.png")
        self.hslogo_LBL.setPixmap(self.hslogo_pic)
        self.hslogo_LBL.setGeometry(550,175,195,180)

    def connec_to_additem(self):
        try:
            self.destroy()
            self.openadditm_ui= add_item_ui()
            self.openadditm_ui.show()
        except Exception as e:
            traceback.print_exc()

    def connecto_auction(self):
        try:
            self.destroy()
            self.openproduct_ui=auction_ui()
            self.openproduct_ui.show()
        except Exception as e:
            traceback.print_exc()

    def connecto_wallet(self):
        try:
            self.destroy()
            self.openproduct_ui=wallet_ui()
            self.openproduct_ui.show()
        except Exception as e:
```

```
traceback.print_exc()


    def connectto_products(self):
        try:
            self.destroy()
            self.openprofile=products_()
            self.openprofile.show()
        except Exception as e:
            traceback.print_exc()


    def connecto_profile(self):
        try:
            self.destroy()
            self.opensettings=profile_ui()
            self.opensettings.show()
        except Exception as e:
            traceback.print_exc()
    def backtologin(self):
        try:
            reply=QMessageBox.question(self, 'Logout Confirmation',
                                       'Are you sure you want to log out?',
                                       QMessageBox.Yes | QMessageBox.No,
QMessageBox.No)
            if reply==QMessageBox.Yes:
                self.destroy()
                self.oplogin=login()
                self.oplogin.show()
```

```
    except Exception as e:
        traceback.print_exc()
```

```
class auction_ui(QDialog):
    def __init__(self):
        super(auction_ui,self).__init__()
        self.setWindowTitle("Auction")
        self.setGeometry(275,140,800,500)

        self.tabsbg_LBL = QLabel(self)
        self.tabsbg_LBL.setStyleSheet("background-
color:rgba(0,12,95,205)")
        self.tabsbg_LBL.setGeometry(0, 0, 791, 80)

        self.LBL_font = QFont()
        self.LBL_font.setFamily(u"Palatino Linotype")
        self.LBL_font.setPointSize(14)
        self.LBL_font.setBold(True)
        self.LBL_font.setItalic(True)
        self.LBL_font.setWeight(75)

        self.LEfont = QtGui.QFont()
        self.LEfont.setPointSize(12)
        # self.LEfont.setBold(True)
        self.LEfont.setItalic(True)
        self.LEfont.setFamily(u"Platino Linotype")

        self.heading_LBLfont = QFont()
        self.heading_LBLfont.setFamily(u"Palatino Linotype")
        self.heading_LBLfont.setPointSize(22)
        self.heading_LBLfont.setBold(True)
```

```
        self.heading_LBLfont.setItalic(True)
        self.heading_LBLfont.setWeight(75)

        '''self.scrollArea = QScrollArea(self)
        self.scrollArea.setObjectName(u"scrollArea")
        self.scrollArea.setGeometry(QRect(5, 100, 783, 380))
        self.scrollArea.setWidgetResizable(True)

        self.scrollAreaWidgetContents = QWidget()

self.scrollAreaWidgetContents.setObjectName(u"scrollAreaWidgetContents")
        self.scrollAreaWidgetContents.setGeometry(QRect(0, 0, 759, 359))

        self.verticalScrollBar = QScrollBar(self.scrollAreaWidgetContents)
        self.verticalScrollBar.setObjectName(u"verticalScrollBar")
        self.verticalScrollBar.setGeometry(QRect(760, 0, 16, 362))
        self.verticalScrollBar.setOrientation(Qt.Vertical)

        self.scrollArea.setWidget(self.scrollAreaWidgetContents)'''

        self.auctionheading_LBL = QLabel(self)
        self.auctionheading_LBL.setText("Auction")
        self.auctionheading_LBL.setGeometry(350, 20, 100, 31)
        self.auctionheading_LBL.setFont(self.heading_LBLfont)
        self.auctionheading_LBL.setStyleSheet("background-color: rgba(255,
255, 255, 10);color : white")

        self.prod_img = QLabel(self)
        self.prod_img.setGeometry(5, 110, 100, 100)
```

```
self.prod_img.setStyleSheet("border: 1px solid black;")


self.prod_idLBL = QLabel(self)
self.prod_idLBL.setText("Product Id:")
self.prod_idLBL.setFont(self.LBL_font)
self.prod_idLBL.setGeometry(225, 120, 100, 30)


self.prod_idLE = QLineEdit(self)
self.prod_idLE.setPlaceholderText("ID")
self.prod_idLE.setFont(self.LEfont)
self.prod_idLE.setGeometry(330, 125, 150, 25)
self.prod_idLE.editingFinished.connect(self.gettingprodinfo)


self.product_name = QLabel(self)
self.product_name.setText("Product Name:-")
self.product_name.setFont(self.LBL_font)
self.product_name.setGeometry(195, 170, 130, 30)


self.productname_LE = QLineEdit(self)
self.productname_LE.setPlaceholderText("Product Name")
self.productname_LE.setFont(self.LEfont)
self.productname_LE.setGeometry(330, 175, 400, 25)


self.due_date = QLabel(self)
self.due_date.setText("Due date:")
self.due_date.setFont(self.LBL_font)
self.due_date.setGeometry(195, 225, 90, 30)
```

```
        self.duedate_LE = QLineEdit(self)
        self.duedate_LE.setFont(self.LEfont)
        self.duedate_LE.setPlaceholderText("Due date")
        self.duedate_LE.setGeometry(300, 225, 220, 25)
        # self.duedate_LE.editingFinished().connect()

        self.prod_descLBL = QLabel(self)
        self.prod_descLBL.setText("Product Description")
        self.prod_descLBL.setWordWrap(True)
        self.prod_descLBL.setFont(self.LBL_font)
        self.prod_descLBL.setGeometry(195, 270, 150, 50)

        self.prod_descLE = QTextEdit(self)
        self.prod_descLE.setPlaceholderText("Description")
        self.prod_descLE.setFont(self.LEfont)
        self.prod_descLE.setGeometry(300, 275, 400, 50)
        self.prod_descLE.installEventFilter(self)

        self.baseprice_LBL = QLabel(self)
        self.baseprice_LBL.setFont(self.LBL_font)
        self.baseprice_LBL.setText("Base Price")
        self.baseprice_LBL.setGeometry(195, 340, 90, 30)

        self.baseprice_LE = QLineEdit(self)
        self.baseprice_LE.setPlaceholderText("Base Price")
        self.baseprice_LE.setFont(self.LEfont)
        self.baseprice_LE.setGeometry(300, 345, 100, 25)
```

```
        self.startauc_BTN = QPushButton("Start Auction", self)
        self.startauc_BTN.setGeometry(240, 390, 150, 30)
        self.startauc_BTN.setFont(self.LBL_font)
        self.startauc_BTN.clicked.connect(self.mainauctionfn)


        self.back_BTN=QPushButton(self)
        self.back_BTN.setText("Back")
        self.back_BTN.setFont(self.LBL_font)
        self.back_BTN.setGeometry(700,0,100,50)
        self.back_BTN.clicked.connect(self.going_back)
        self.back_BTN.setStyleSheet("background-color: rgba(255, 255, 255,
10);color : white")


    def gettingprodinfo(self):
        self.currentauc_details={}
        prod_id=self.prod_idLE.text()
        try:
            if prod_id:
                connection = sql.connect(host="192.168.102.206",
user="root", password="sairam", database="aucsof_ip")
                main_cursor = connection.cursor()


                query=f"select
product,last_date,product_desc,base_price,bid_increement,product_id from
products where product_id={prod_id}"
                query1=f"select image from images where img_id={prod_id}"
                main_cursor.execute(query)
                self.result = main_cursor.fetchall()
                main_cursor.execute(query1)
                data=main_cursor.fetchone()
```

```
                if data is not None:
                    image_data = data[0]
                    self.temp = QPixmap()
                    self.temp.loadFromData(image_data)
                    # print(self.temp)
                    self.resized = self.temp.scaled(100, 100,
QtCore.Qt.KeepAspectRatio)
                    self.prod_img.setPixmap(self.resized)
                self.productname_LE.setText(self.result[0][0])
                self.currentauc_details["Product
Name"]=str(self.result[0][0])
                self.productname_LE.setEnabled(False)
                self.duedate_LE.setText(str(self.result[0][1]))
                self.currentauc_details["Due Date"]=str(self.result[0][1])
                self.duedate_LE.setEnabled(False)
                self.prod_descLE.setText(self.result[0][2])
                self.currentauc_details["Product
Description"]=str(self.result[0][2])
                self.prod_descLE.setEnabled(False)
                self.baseprice_LE.setText(self.result[0][3])
                self.currentauc_details["Base Price"] =
str(self.result[0][3])
                self.baseprice_LE.setEnabled(False)
                self.currentauc_details["Bid
Increement"]=str(self.result[0][4])
                self.currentauc_details["Product
Id"]=str(self.result[0][5])
                #self.currentauc_details["Imagebindata"]=image_data
            else:
                pass
```

```
        except Exception as e:
            traceback.print_exc()


    def mainauctionfn(self):
        try:
            reply = QMessageBox.question(self, 'Auction Confirmation',
                                         'Are you sure you want to enter
the auction?',
                                         QMessageBox.Yes | QMessageBox.No,
QMessageBox.No)


            if self.prod_idLE.text()=="":
                QtWidgets.QMessageBox.warning(self,"Error","Please Fill in
the details")
            elif reply==QMessageBox.Yes:
                print("Starting auction server...")
                self.close()


                self.server_window =
AuctionServer(self.currentauc_details)
                self.server_window.show()
            else:
                pass
        except Exception as ex:
            traceback.print_exc()
    def going_back(self):
        try:
            self.destroy()
            self.main_uiobj=main_ui()
            self.main_uiobj.show()
```

```
        except Exception as e:
            traceback.print_exc()


    def eventFilter(self, source, event):
        try:
            if event.type() == event.Enter and source == self.prod_descLE:
                if self.prod_descLE.toPlainText()=="":
                    pass
                else:
                    size = self.prod_descLE.document().size()
                    self.prod_descLE.setFixedSize(QSize(int(size.width())
+ 20, int(size.height() + 20)))
                    self.prod_descLE.raise_()
            elif event.type() == event.Leave and source ==
self.prod_descLE:
                self.prod_descLE.setFixedSize( 400, 50)
            return super(auction_ui, self).eventFilter(source, event)
        except Exception as e:
            traceback.print_exc()
```

```
class AuctionServer(QDialog):
    def __init__(self, currentauc_details):
        super().__init__()
        self.setWindowTitle("Auction Server")
        self.showMaximized()
        self.setStyleSheet("""
            background-color: #f4f6f9;
            font-family: 'Arial';
            color: #333;
        """)

        # Initialize live auction details
        self.liveaucdetails = currentauc_details
        self.starttime = time.time()

        if self.liveaucdetails:
            self.liveaucdetails["Minimum Next Bid"] =
int(self.liveaucdetails['Base Price']) + int(
                self.liveaucdetails['Bid Increement'])

        self.HOST = '192.168.102.206'
        self.PORT = 55444
        self.clients = []

        # Chat box and UI elements
        self.chat_box = QTextEdit(self)
        self.chat_box.setReadOnly(True)
        self.chat_box.setFont(QFont("Arial", 12))
```

```
        self.chat_box.setStyleSheet("""
            background-color: #ffffff;
            color: #333;
            border: 1px solid #ccc;
            border-radius: 5px;
            padding: 10px;
        """)

        self.message_bar = QLineEdit(self)
        self.message_bar.setFont(QFont("Arial", 12))
        self.message_bar.setPlaceholderText("Type your message here...")
        self.message_bar.setStyleSheet("""
            border: 1px solid #ccc;
            border-radius: 5px;
            padding: 10px;
        """)
        self.message_bar.returnPressed.connect(self.send_message)

        self.start_button = QPushButton("Start Auction", self)
        self.start_button.setStyleSheet("""
            background-color: #007bff;
            color: white;
            padding: 10px 20px;
            border: none;
            border-radius: 5px;
            font-size: 16px;
            font-weight: bold;
        """)
```

```
        self.start_button.setCursor(Qt.PointingHandCursor)
        self.start_button.clicked.connect(self.start_auction)

        self.product_name_label = QLabel(f"Product Name:
{self.liveaucdetails['Product Name']}", self)
        self.product_name_label.setFont(QFont("Arial", 18, QFont.Bold))

        self.product_desc_label = QLabel(f"Description:
{self.liveaucdetails['Product Description']}", self)
        self.product_desc_label.setWordWrap(True)
        self.product_desc_label.setFont(QFont("Arial", 16,QFont.Bold))

        self.base_price_label = QLabel(f"Base Price:
${self.liveaucdetails['Base Price']}", self)
        self.base_price_label.setFont(QFont("Arial", 16,QFont.Bold))

        self.min_next_bid_label = QLabel(f"Minimum Next Bid:
${self.liveaucdetails['Minimum Next Bid']}", self)
        self.min_next_bid_label.setFont(QFont("Arial", 16,QFont.Bold))

        self.auction_status_label = QLabel("Auction Status: Not Started",
self)
        self.auction_status_label.setFont(QFont("Arial", 14, QFont.Bold))
        self.auction_status_label.setStyleSheet("color: #555;")

        # Layout
        layout = QVBoxLayout()
        layout.setSpacing(20)
        layout.addWidget(QLabel("<h1 style='color:#007bff;'>Auction Server
Running</h1>", self),
```

```
                        alignment=Qt.AlignCenter)
        layout.addWidget(self.product_name_label)
        layout.addWidget(self.product_desc_label)
        layout.addWidget(self.base_price_label)
        layout.addWidget(self.min_next_bid_label)
        layout.addWidget(self.auction_status_label)
        layout.addWidget(self.chat_box, stretch=2)
        layout.addWidget(self.message_bar)
        layout.addWidget(self.start_button, alignment=Qt.AlignRight)
        self.setLayout(layout)

        # Start server in a separate thread
        server_thread = threading.Thread(target=self.start_server)
        server_thread.daemon = True
        server_thread.start()

    def start_server(self):
        self.serversock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.serversock.bind((self.HOST, self.PORT))
        self.serversock.listen(3)
        self.update_chat_display(f"[LISTENING] Server is listening on
{self.HOST}:{self.PORT}")

        while True:
            self.clientsock, self.address = self.serversock.accept()
            self.clients.append(self.clientsock)
            self.update_chat_display(f"[NEW CONNECTION] {self.address}
connected.")
```

```
            client_thread = threading.Thread(target=self.handle_client,
args=(self.clientsock,))
            client_thread.daemon = True
            client_thread.start()



    def handle_client(self, clientsock):
        try:
            clientsock.send(
                "Welcome to the Auction! Type 'details' for auction info.
Type 'exit' to leave.".encode('utf-8')
            )
            while True:
                self.receive_messages(clientsock)
        except Exception as e:
            print(f"Error in client handler: {e}")
        finally:
            self.remove_client(clientsock)




    def remove_client(self, clientsock):
        """Remove the client and close the socket."""
        if clientsock in self.clients:
            self.clients.remove(clientsock)
            clientsock.close()
            self.update_chat_display("A client has disconnected.")
```

```
    def receive_messages(self, clientsock):
        try:
            message = clientsock.recv(1024).decode('utf-8')
            print(message)
            if not message:
                # Empty message indicates client disconnected
                self.update_chat_display(f"[DISCONNECT]
{clientsock.getpeername()} disconnected.")
                self.remove_client(clientsock)
                return

            # Process the message
            if message.strip().lower() == "exit":
                self.update_chat_display(f"[CLIENT EXIT]
{clientsock.getpeername()} exited the auction.")
                clientsock.send("DISCONNECT".encode('utf-8'))
                self.remove_client(clientsock)
            elif message.strip().lower() == "details":
                if self.liveaucdetails:
                    sending_data = pickle.dumps(self.liveaucdetails)
                    clientsock.send(b"PICKLE" + sending_data)
                    self.update_chat_display(f"Auction details sent to
{clientsock.getpeername()}.")
            else:
                self.update_chat_display(f"[CLIENT MESSAGE]
{clientsock.getpeername()} says: {message}")
                self.broadcast_message(f"Client {clientsock.getpeername()}
says: {message}", clientsock)
        except Exception as e:
            print(f"Error receiving message: {e}")
```

```
            self.remove_client(clientsock)



    def update_chat_display(self, message):
        self.chat_box.append(message)


    def send_message(self):
        message = self.message_bar.text().strip()
        if message:
            self.update_chat_display(f"[SERVER] {message}")
            self.broadcast_message(message)
            self.message_bar.clear()


    def broadcast_message(self, message, sender_socket=None):
        for client in self.clients:
            if client != sender_socket:
                try:
                    client.send(message.encode('utf-8'))
                except Exception as e:
                    print("Failed to send message to a client", e)


    def start_auction(self):
        # Start the auction and notify clients
        if self.liveaucdetails:
            self.liveaucdetails["Auction Started"] = True
            self.auction_status_label.setText("Auction Status: Active")
            self.broadcast_message(f"Auction for
{self.liveaucdetails['Product Name']} has started!")
```

```
            self.update_chat_display(f"Auction for
{self.liveaucdetails['Product Name']} has started!")


 class products_(QDialog):
    def __init__(self):
        super(products_, self).__init__()
        self.setWindowTitle("Products")
        self.setGeometry(275,140,800,500)


        self.desc=[]
        self.readm=[]


        self.tabsbg_LBL = QLabel(self)
        self.tabsbg_LBL.setStyleSheet("background-
color:rgba(0,12,95,205)")
        self.tabsbg_LBL.setGeometry(0, 0, 791, 80)


        self.LBL_font = QFont()
        self.LBL_font.setFamily("Palatino Linotype")
        self.LBL_font.setPointSize(14)
        self.LBL_font.setBold(True)
        self.LBL_font.setItalic(True)
        self.LBL_font.setWeight(75)


        self.txt_font = QFont()
        self.txt_font.setFamily("Palatino Linotype")
        self.txt_font.setPointSize(12)
        self.txt_font.setBold(True)
        self.txt_font.setItalic(True)
```

```
        self.txt_font.setWeight(75)


        self.heading_LBLfont = QFont()
        self.heading_LBLfont.setFamily("Palatino Linotype")
        self.heading_LBLfont.setPointSize(22)
        self.heading_LBLfont.setBold(True)
        self.heading_LBLfont.setItalic(True)
        self.heading_LBLfont.setWeight(75)


        # Setting up the scroll area
        self.scrollArea = QScrollArea(self)
        self.scrollArea.setGeometry(QRect(20, 120, 761, 361))
        self.scrollArea.setWidgetResizable(True)


        # Creating a widget to hold the scroll area's contents
        self.scrollAreaWidgetContents = QWidget()
        self.scrollArea.setWidget(self.scrollAreaWidgetContents)


        try:
            connection = sql.connect(host="192.168.102.206", user="root",
password="sairam", database="aucsof_ip")
            main_cursor = connection.cursor()


            self.desc_query = "SELECT product, product_desc FROM products
order by product_id;"
            main_cursor.execute(self.desc_query)
            self.desc = main_cursor.fetchall()


            b, j = 10, 130
```

```
            max_height = 0

            for i in range(1, len(self.desc) + 1):
                prodname = self.desc[i - 1][0]
                proddesc = self.desc[i - 1][1]
                totalinf = f"{prodname}\n{proddesc}"

                self.prod_img = QLabel(self.scrollAreaWidgetContents)
                self.prod_img.setGeometry(5, b, 100, 100)
                self.prod_img.setStyleSheet("border: 1px solid black;")

                query1 = """select image from images where img_id =""" +
str(i)
                #print(query1)

                try:
                    main_cursor.execute(query1)
                    data = main_cursor.fetchone()

                    if data is not None:
                        image_data = data[0]
                        self.temp=QPixmap()
                        self.temp.loadFromData(image_data)
                        #print(self.temp)
                        self.resized = self.temp.scaled(100, 100,
QtCore.Qt.KeepAspectRatio)
                        self.prod_img.setPixmap(self.resized)

                    else:
```

```
                        self.prod_img.setText("Image not found.")


                except Exception as e:
                    #print(f"Error fetching image for product {prodname}:
{img_err}")
                    traceback.print_exc()
                #self.prod_img.setPixmap(pixmap)



                self.product_desc = QLabel(self.scrollAreaWidgetContents)
                self.product_desc.setFont(self.txt_font)
                self.product_desc.setText(totalinf)
                self.product_desc.setWordWrap(True)
                self.product_desc.setGeometry(120, b + 10, 500, 80)
                self.desc.append(self.product_desc)


                self.readmore_BTN =
QPushButton(self.scrollAreaWidgetContents)
                self.readmore_BTN.setFont(self.txt_font)
                self.readmore_BTN.setText("Read More")
                self.readmore_BTN.setGeometry(650, b + 60, 90, 25)
                self.readmore_BTN.clicked.connect(self.openingreadmore)
                self.readm.append(self.readmore_BTN)


                b += 130
                j += 135
                max_height = j + 20


            self.scrollAreaWidgetContents.setMinimumHeight(max_height)
```

```
        except Exception as err:
            print(f"Database Error: {err}")
            traceback.print_exc()
        finally:
            if connection:
                connection.close()


        self.productsheading_LBL = QLabel(self)
        self.productsheading_LBL.setText("Products")
        self.productsheading_LBL.setGeometry(350, 20, 120, 31)
        self.productsheading_LBL.setFont(self.heading_LBLfont)
        self.productsheading_LBL.setStyleSheet("background-color:
rgba(255, 255, 255, 10); color: white")


        self.back_BTN = QPushButton(self)
        self.back_BTN.setText("Back")
        self.back_BTN.setFont(self.LBL_font)
        self.back_BTN.setGeometry(700, 0, 100, 50)
        self.back_BTN.clicked.connect(self.going_back)
        self.back_BTN.setStyleSheet("background-color: rgba(255, 255, 255,
10); color: white")


    def openingreadmore(self):
        try:
            self.descreadm=self.sender()
            retrivindesc=self.readm.index(self.descreadm)+1
            print(self.descreadm)
            print(retrivindesc)
```

```
            self.destroy()
            self.op=ReadMore(retrivindesc)
            self.op.show()
        except Exception as e:
            traceback.print_exc()


    def going_back(self):
        try:
            self.close()
            self.main_uiobj = main_ui()
            self.main_uiobj.show()
        except Exception as e:
            traceback.print_exc()
```

```
class ReadMore(QWidget):
    def __init__(self, retrieving_desc):
        super(ReadMore, self).__init__()
        self.setWindowTitle("Read More About the Product")
        self.setGeometry(275, 140, 800, 500)

        # Fonts
        self.LBL_font = QFont("Palatino Linotype", 14, QFont.Bold,
italic=True)
        self.txt_font = QFont("Palatino Linotype", 12, QFont.Bold,
italic=True)
        self.heading_LBLfont = QFont("Palatino Linotype", 22, QFont.Bold,
italic=True)

        # Background for header
        self.tabsbg_LBL = QLabel(self)
        self.tabsbg_LBL.setStyleSheet("background-
color:rgba(0,12,95,205)")
        self.tabsbg_LBL.setGeometry(0, 0, 791, 80)

        # Back button
        self.back_BTN = QPushButton(self)
        self.back_BTN.setText("Back")
        self.back_BTN.setFont(self.LBL_font)
        self.back_BTN.setGeometry(700, 0, 100, 50)
        self.back_BTN.clicked.connect(self.going_back)
```

```
        self.back_BTN.setStyleSheet("background-color: rgba(255, 255, 255,
10); color: white")


        try:
            # Database connection and query execution
            connection = sql.connect(
                host="192.168.102.206", user="root", password="sairam",
database="aucsof_ip"
            )
            main_cursor = connection.cursor()
            query = f"SELECT product, product_desc, base_price,
last_date,bid_increement FROM products  WHERE product_id={retrieving_desc}
ORDER BY product_id"
            main_cursor.execute(query)
            self.testresult=main_cursor.fetchall()
            self.result={}



            if self.testresult:
                self.prodhead = QLabel(self)
                self.prodhead.setFont(self.heading_LBLfont)
                self.prodhead.setText(self.testresult[0][0])
                self.result["Product Name"]=str(self.testresult[0][0])
                self.prodhead.setGeometry(10, 10, 700, 50)
                self.prodhead.setStyleSheet("color: white;")

                # Product description
                self.desc_label = QLabel(self)
                self.desc_label.setFont(self.txt_font)
                self.desc_label.setWordWrap(True)
```

```
                self.desc_label.setText(f"Description:
{self.testresult[0][1]}")
                self.result["Product
Description"]=str(self.testresult[0][1])
                self.desc_label.setGeometry(10, 100, 450, 150)
                self.desc_label.setStyleSheet("color: black;")


                # Base price
                self.price_label = QLabel(self)
                self.price_label.setFont(self.txt_font)
                self.price_label.setText(f"Base Price:
${self.testresult[0][2]}")
                self.result["Base Price"]=str(self.testresult[0][2])
                self.price_label.setGeometry(10, 250, 300, 40)
                self.price_label.setStyleSheet("color: black;")


                self.minnextbid=QLabel(f"Minimum Next Bid:
{int(self.testresult[0][2])+1000}" ,self)
                self.minnextbid.setFont(self.LBL_font)
                self.minnextbid.setGeometry(10,360,300,40)


                # Due date
                self.due_date_label = QLabel(self)
                self.due_date_label.setFont(self.txt_font)
                self.due_date_label.setText(f"Due Date:
{self.testresult[0][3]}")
                self.result["Due Date"]=str(self.testresult[0][3])
                self.due_date_label.setGeometry(10, 300, 300, 40)
                self.due_date_label.setStyleSheet("color: black;")
                self.result["Bid Increement"] = str(self.testresult[0][4])
```

```
                self.result["Product Id"]=retrieving_desc


                # Product Image
                self.product_image=QLabel(self)
                q1=f"select image from images where img_id
={retrieving_desc}"
                main_cursor.execute(q1)
                result=main_cursor.fetchall()
                if result is not None:
                    self.image_data = result[0][0]
                    self.temp = QPixmap()
                    self.temp.loadFromData(self.image_data)
                    # print(self.temp)
                    self.resized = self.temp.scaled(250, 250,
QtCore.Qt.KeepAspectRatio)
                    self.product_image.setPixmap(self.resized)
                    self.product_image.setGeometry(500,150,250,250)
                else:
                    self.product_image.setText("Image Not Available")
                    self.product_image.setStyleSheet("color: red;")
                    self.product_image.setGeometry(500, 100, 250, 250)


                # Start Auction button
                self.start_auction_btn = QPushButton("Start Auction",
self)
                self.start_auction_btn.setFont(self.LBL_font)
                self.start_auction_btn.setGeometry(300, 400, 200, 50)
                self.start_auction_btn.setStyleSheet("background-color:
green; color: white;")
                self.start_auction_btn.clicked.connect(self.start_auction)
```

```
        except sql.Error as err:
            print(f"Database Error: {err}")

    def going_back(self):
        try:
            self.close()
            self.products_uiobj = products_()
            self.products_uiobj.show()
        except Exception as e:
            traceback.print_exc()

    def start_auction(self):
        # Logic to start auction
        try:
            reply = QMessageBox.question(self, 'Auction Confirmation',
                                         'Are you sure you want to enter
the auction?',
                                         QMessageBox.Yes | QMessageBox.No,
QMessageBox.No)
            if reply==QMessageBox.Yes:
                print("Auction is Starting for the product!")
                print("Starting auction server...")
                self.close()
                #self.result["Imagebindata"]=self.image_data
                self.server_window2 = AuctionServer(self.result)
                self.server_window2.show()
        except Exception as e:
            traceback.print_exc()
```

```
class add_item_ui(QDialog):
    def __init__(self):
        super(add_item_ui,self).__init__()
        self.setWindowTitle("Add Items")
        self.setGeometry(275,140,800,500)

        self.tabsbg_LBL = QLabel(self)
        self.tabsbg_LBL.setStyleSheet("background-
color:rgba(0,12,95,205)")
        self.tabsbg_LBL.setGeometry(0, 0, 791, 80)

        self.LBL_font = QFont()
        self.LBL_font.setFamily(u"Palatino Linotype")
        self.LBL_font.setPointSize(14)
        self.LBL_font.setBold(True)
        self.LBL_font.setItalic(True)
        self.LBL_font.setWeight(75)

        self.LEfont = QtGui.QFont()
        self.LEfont.setPointSize(12)
        # self.LEfont.setBold(True)
        self.LEfont.setItalic(True)
        self.LEfont.setFamily(u"Platino Linotype")

        self.heading_LBLfont = QFont()
        self.heading_LBLfont.setFamily(u"Palatino Linotype")
```

```
        self.heading_LBLfont.setPointSize(22)
        self.heading_LBLfont.setBold(True)
        self.heading_LBLfont.setItalic(True)
        self.heading_LBLfont.setWeight(75)

        self.addtitm_uiheading_LBL = QLabel(self)
        self.addtitm_uiheading_LBL.setText("Add Items")
        self.addtitm_uiheading_LBL.setGeometry(340, 20, 133, 31)
        self.addtitm_uiheading_LBL.setFont(self.heading_LBLfont)
        self.addtitm_uiheading_LBL.setStyleSheet("background-color:
rgba(255, 255, 255, 10);color : white")

        self.prod_img = QLabel(self)
        self.prod_img.setGeometry(5, 110, 110, 110)
        self.prod_img.setStyleSheet("border: 1px solid black;")

        self.addimg_BTN=QPushButton(self)
        self.addimg_BTN.setText("Add Image")
        self.addimg_BTN.clicked.connect(self.openingfiledialogue)
        self.addimg_BTN.setGeometry(15,225,75,25)

        self.prod_idLBL=QLabel(self)
        self.prod_idLBL.setText("Product Id:")
        self.prod_idLBL.setFont(self.LBL_font)
        self.prod_idLBL.setGeometry(225,120,100,30)

        self.prod_idLE = QLineEdit(self)
        self.prod_idLE.setPlaceholderText("New ID")
        self.prod_idLE.setFont(self.LEfont)
```

```
        self.prod_idLE.setGeometry(330, 125, 150, 25)
        try:
            connection = sql.connect(host="192.168.102.206", user="root",
password="sairam", database="aucsof_ip")
            main_cursor = connection.cursor()
            q1="select max(product_id) from products;"
            main_cursor.execute(q1)
            temp=main_cursor.fetchall()
            print(temp[0][0])
            self.prod_idLE.setText(str(temp[0][0]+1))
            self.prod_idLE.setEnabled(False)
        except Exception as e:
            traceback.print_exc()

        self.product_name=QLabel(self)
        self.product_name.setText("Product Name:-")
        self.product_name.setFont(self.LBL_font)
        self.product_name.setGeometry(195,170,130,30)

        self.productname_LE = QLineEdit(self)
        self.productname_LE.setPlaceholderText("Product Name")
        self.productname_LE.setFont(self.LEfont)
        self.productname_LE.setGeometry(330, 175, 150, 25)

        self.due_date=QLabel(self)
        self.due_date.setText("Due date:")
        self.due_date.setFont(self.LBL_font)
        self.due_date.setGeometry(195,225,90,30)
```

```
        #self.duewid_LE=QLineEdit(self)
        #self.duewid_LE.setFont(self.LEfont)
        #self.duewid_LE.setGeometry(520,225,220,150)

        self.duedate_LE=QDateEdit(QDate.currentDate(),self)
        self.duedate_LE.setCalendarPopup(True)
        self.duedate_LE.setDisplayFormat("yyyy-MM-dd")
        self.duedate_LE.setFont(self.LEfont)
        self.duedate_LE.setGeometry(300,225,220,25)
        #self.duedate_LE.installEventFilter(self)

        self.due_datewid=QCalendarWidget(self)
        #self.due_datewid.resize(self.duedate_LE.size())
        self.due_datewid.hide()
        self.due_datewid.clicked.connect(self.getdate)

        self.prod_descLBL=QLabel(self)
        self.prod_descLBL.setText("Product Description")
        self.prod_descLBL.setWordWrap(True)
        self.prod_descLBL.setFont(self.LBL_font)
        self.prod_descLBL.setGeometry(195,270,150,50)

        self.prod_descLE = QTextEdit(self)
        self.prod_descLE.setPlaceholderText("Enter your description
here.... Min 30 words")
        self.prod_descLE.setFont(self.LEfont)
        self.prod_descLE.setGeometry(300, 275, 220, 50)

        self.baseprice_LBL=QLabel(self)
```

```
        self.baseprice_LBL.setFont(self.LBL_font)
        self.baseprice_LBL.setText("Base Price")
        self.baseprice_LBL.setGeometry(195,340,90,30)

        self.baseprice_LE = QLineEdit(self)
        self.baseprice_LE.setPlaceholderText("Base Price")
        self.baseprice_LE.setFont(self.LEfont)
        self.baseprice_LE.setGeometry(300, 345, 100, 25)

        self.additm_BTN=QPushButton(self)
        self.additm_BTN.setFont(self.LBL_font)
        self.additm_BTN.setText("Add item")
        self.additm_BTN.setGeometry(240,390,100,30)
        self.additm_BTN.clicked.connect(self.addproduct)

        self.back_BTN = QPushButton(self)
        self.back_BTN.setText("Back")
        self.back_BTN.setFont(self.LBL_font)
        self.back_BTN.setGeometry(700, 0, 100, 50)
        self.back_BTN.clicked.connect(self.going_back)
        self.back_BTN.setStyleSheet("background-color: rgba(255, 255, 255,
10);color : white")

        self.remove_itm_BTN = QPushButton(self)
        self.remove_itm_BTN.setText("Delete Items")
        self.remove_itm_BTN.setFont(self.LBL_font)
        self.remove_itm_BTN.setGeometry(590, 80, 200, 35)
        self.remove_itm_BTN.clicked.connect(self.rem_ui)
```

```
    '''def eventFilter(self, source, event):
        try:
            if event.type() == event.Enter and source == self.duedate_LE:
                print("in")
                self.due_datewid.setGeometry(300,250,310,180)
                self.due_datewid.show()
                self.due_datewid.raise_()
                #self.due_datewid.setGeometry(530,225,)
            elif event.type() == event.Leave and source ==
self.duedate_LE:
                print("Not in")
                self.due_datewid.hide()
            return super(add_item_ui, self).eventFilter(source, event)
        except Exception as e:
            traceback.print_exc()'''

    def getdate(self):
        self.getdatefrmwid = self.due_datewid.selectedDate().toPyDate()
        try:
            self.duedate_LE.setText(str(self.getdatefrmwid))
            print(self.getdatefrmwid)
        except Exception as e:
            traceback.print_exc()

    def openingfiledialogue(self):
        options = QFileDialog.Options()
        fileName,_ = QFileDialog.getOpenFileName(self, "Open Image File",
"",
```

```
                                                    "Images (*.png *.xpm
*.jpg);;All Files (*)", options=options)


        if fileName:

            try:

                self.image_path = fileName

                print(self.image_path)

                self.pixmapimg_prod = QPixmap(fileName)

self.prod_img.setPixmap(self.pixmapimg_prod.scaled(self.prod_img.size(),
aspectRatioMode=True))

            except Exception as e:

                traceback.print_exc()


    def addproduct(self):

        try:

            product_id = int(self.prod_idLE.text())

            productname = self.productname_LE.text()

            lastdate = self.duedate_LE.text()

            product_desc = self.prod_descLE.toPlainText()

            baseprice = self.baseprice_LE.text()


            '''if not all([product_id, productname, lastdate,
product_desc, baseprice, self.image_path]):

                QtWidgets.QMessageBox.warning(self, 'Error', 'All fields
must be filled, including the image.')

                return'''


            connection = sql.connect(host="192.168.102.206", user="root",
password="sairam", database="aucsof_ip")
```

```
            main_cursor = connection.cursor()


            # Insert image into the images table and get the image_id
            with open(self.image_path, 'rb') as file:
                img_data = file.read()
            addprod_query = """
                                INSERT INTO products (product_id, product,
last_date, product_desc, status, base_price)
                                VALUES (%s, %s, %s, %s, %s, %s)
                            """
            print(addprod_query)
            main_cursor.execute(addprod_query, (product_id, productname,
lastdate, product_desc, "unsold", baseprice))



            insert_image_query = "INSERT INTO images (img_id,image) VALUES
(%s,%s)"
            print(self.image_path)
            main_cursor.execute(insert_image_query, (product_id,img_data))
            image_id = main_cursor.lastrowid  # Get the last inserted
image_id



            connection.commit()


            QtWidgets.QMessageBox.information(self, 'Success', 'Item added
successfully.')
            self.productname_LE.clear()
            self.duedate_LE.clear()
            self.prod_img.clear()
```

```
            self.prod_descLE.clear()
            self.baseprice_LE.clear()
        except Exception as e:
            traceback.print_exc()
            QtWidgets.QMessageBox.information(self, 'Error 404', 'Item
could not be added!.')
        finally:
            connection.close()


    def rem_ui(self):
        try:
            self.close()
            self.removeitmobj = removeitem()
            self.removeitmobj.exec_()
        except Exception as e:
            traceback.print_exc()



    def going_back(self):
        try:
            self.close()
            self.main_uiobj = main_ui()
            self.main_uiobj.show()
        except Exception as e:
            traceback.print_exc()
```

```
class removeitem(QDialog):
    def __init__(self):
        super(removeitem, self).__init__()
        self.setWindowTitle("Remove Item")
        self.setGeometry(275, 140, 800, 500)

        # UI Components Setup
        self.tabsbg_LBL = QLabel(self)
        self.tabsbg_LBL.setStyleSheet("background-
color:rgba(0,12,95,205)")
        self.tabsbg_LBL.setGeometry(0, 0, 791, 80)

        self.LBL_font = QFont()
        self.LBL_font.setFamily(u"Palatino Linotype")
        self.LBL_font.setPointSize(14)
        self.LBL_font.setBold(True)
        self.LBL_font.setItalic(True)
        self.LBL_font.setWeight(75)

        self.heading_LBLfont = QFont()
        self.heading_LBLfont.setFamily(u"Palatino Linotype")
        self.heading_LBLfont.setPointSize(22)
        self.heading_LBLfont.setBold(True)
        self.heading_LBLfont.setItalic(True)
        self.heading_LBLfont.setWeight(75)
```

```
        self.LEfont = QFont()
        self.LEfont.setPointSize(12)
        self.LEfont.setItalic(True)
        self.LEfont.setFamily(u"Platino Linotype")

        # UI Elements
        self.prod_img = QLabel(self)
        self.prod_img.setGeometry(5, 110, 100, 100)
        self.prod_img.setStyleSheet("border: 1px solid black;")

        self.prod_idLBL = QLabel(self)
        self.prod_idLBL.setText("Product Id:")
        self.prod_idLBL.setFont(self.LBL_font)
        self.prod_idLBL.setGeometry(225, 120, 100, 30)

        self.prod_idLE = QLineEdit(self)
        self.prod_idLE.setPlaceholderText("ID")
        self.prod_idLE.setFont(self.LEfont)
        self.prod_idLE.setGeometry(330, 125, 150, 25)
        self.prod_idLE.editingFinished.connect(self.gettingprodinfo)

        self.product_name = QLabel(self)
        self.product_name.setText("Product Name:")
        self.product_name.setFont(self.LBL_font)
        self.product_name.setGeometry(195, 170, 130, 30)

        self.productname_LE = QLineEdit(self)
        self.productname_LE.setPlaceholderText("Product Name")
```

```
self.productname_LE.setFont(self.LEfont)
self.productname_LE.setGeometry(330, 175, 400, 25)

self.due_date = QLabel(self)
self.due_date.setText("Due date:")
self.due_date.setFont(self.LBL_font)
self.due_date.setGeometry(195, 225, 90, 30)

self.duedate_LE = QLineEdit(self)
self.duedate_LE.setFont(self.LEfont)
self.duedate_LE.setPlaceholderText("Due date")
self.duedate_LE.setGeometry(300, 225, 220, 25)

self.prod_descLBL = QLabel(self)
self.prod_descLBL.setText("Product Description")
self.prod_descLBL.setWordWrap(True)
self.prod_descLBL.setFont(self.LBL_font)
self.prod_descLBL.setGeometry(195, 270, 150, 50)

self.prod_descLE = QTextEdit(self)
self.prod_descLE.setPlaceholderText("Description")
self.prod_descLE.setFont(self.LEfont)
self.prod_descLE.setGeometry(300, 275, 400, 50)

self.baseprice_LBL = QLabel(self)
self.baseprice_LBL.setFont(self.LBL_font)
self.baseprice_LBL.setText("Base Price")
self.baseprice_LBL.setGeometry(195, 340, 90, 30)
```

```
        self.baseprice_LE = QLineEdit(self)
        self.baseprice_LE.setPlaceholderText("Base Price")
        self.baseprice_LE.setFont(self.LEfont)
        self.baseprice_LE.setGeometry(300, 345, 100, 25)

        self.removeitmheading_LBL = QLabel(self)
        self.removeitmheading_LBL.setText("Delete Items")
        self.removeitmheading_LBL.setGeometry(330, 20, 200, 31)
        self.removeitmheading_LBL.setFont(self.heading_LBLfont)
        self.removeitmheading_LBL.setStyleSheet("background-color:
rgba(255, 255, 255, 10);color : white")

        self.delete_BTN = QPushButton(self)
        self.delete_BTN.setText("Delete Item")
        self.delete_BTN.setFont(self.LBL_font)
        self.delete_BTN.setGeometry(300, 400, 150, 50)
        self.delete_BTN.clicked.connect(self.delete_item)
        self.delete_BTN.setStyleSheet("background-color: red; color:
white")

        self.back_BTN = QPushButton(self)
        self.back_BTN.setText("Back")
        self.back_BTN.setFont(self.LBL_font)
        self.back_BTN.setGeometry(700, 0, 100, 50)
        self.back_BTN.clicked.connect(self.going_back)
        self.back_BTN.setStyleSheet("background-color: rgba(255, 255, 255,
10);color : white")
```

```
    def gettingprodinfo(self):
        prod_id = self.prod_idLE.text()
        try:
            connection = sql.connect(host="192.168.102.206", user="root",
password="sairam", database="aucsof_ip")
            main_cursor = connection.cursor()


            query = f"SELECT product, last_date, product_desc, base_price
FROM products WHERE product_id={prod_id}"
            main_cursor.execute(query)
            result = main_cursor.fetchall()
            if result:
                self.productname_LE.setText(result[0][0])
                self.duedate_LE.setText(str(result[0][1]))
                self.prod_descLE.setText(result[0][2])
                self.baseprice_LE.setText(result[0][3])
                self.product_image = QLabel(self)
                q1 = f"select image from images where img_id ={prod_id}"
                main_cursor.execute(q1)
                result2 = main_cursor.fetchall()
                if result2 is not None:
                    self.image_data = result2[0][0]
                    self.temp = QPixmap()
                    self.temp.loadFromData(self.image_data)
                    # print(self.temp)
                    self.resized = self.temp.scaled(100, 100,
QtCore.Qt.KeepAspectRatio)
                    self.prod_img.setPixmap(self.resized)
                    #self.prod_img.setGeometry(500, 150, 250, 250)
```

```
                else:
                    self.prod_img.setText("Image Not Available")
                    self.prod_img.setStyleSheet("color: red;")
                    self.prod_img.setGeometry(500, 100, 250, 250)
            else:
                QtWidgets.QMessageBox.warning(self, 'Error', 'No product
found with the given ID.')
        except Exception as e:
            traceback.print_exc()
        finally:
            connection.close()


    def delete_item(self):
        prod_id = self.prod_idLE.text()
        try:
            connection = sql.connect(host="192.168.102.206", user="root",
password="sairam", database="aucsof_ip")
            main_cursor = connection.cursor()


            delete_query = f"DELETE FROM products WHERE
product_id={prod_id}"
            main_cursor.execute(delete_query)
            del_imgquery = f"delete from images where img_id={prod_id}"
            main_cursor.execute(del_imgquery)
            connection.commit()


            if main_cursor.rowcount > 0:
                QtWidgets.QMessageBox.information(self, 'Success',
f'Product ID {prod_id} deleted successfully!')
                self.clear_fields()
```

```
            else:
                QtWidgets.QMessageBox.warning(self, 'Error', f'No product
found with ID {prod_id}.')
        except Exception as e:
            traceback.print_exc()
        finally:
            connection.close()


    def clear_fields(self):
        self.prod_idLE.clear()
        self.productname_LE.clear()
        self.duedate_LE.clear()
        self.prod_descLE.clear()
        self.baseprice_LE.clear()


    def going_back(self):
        try:
            self.close()
            self.additmobj = add_item_ui()
            self.additmobj.exec_()
        except Exception as e:
            traceback.print_exc()
```

```
class wallet_ui(QWidget):
    def __init__(self):
        super().__init__()
        self.init_ui()
        self.wallet_balance = 0

    def init_ui(self):
        self.setWindowTitle("Auction Wallet System")
        self.setFixedSize(800, 500)

        # Set up fonts and styles
        self.setFont(QFont("Arial", 10))

        self.back_btn = QPushButton("Back")
        self.back_btn.setStyleSheet(
            "padding: 5px 10px; background-color: #FF5722; color: white;
border: none; border-radius: 5px;")
        self.back_btn.clicked.connect(self.go_back)

        # Wallet Setup
        self.wallet_label = QLabel("Create Wallet:")
        self.wallet_label.setFont(QFont("Arial", 12, QFont.Bold))

        self.wallet_input = QLineEdit()
        self.wallet_input.setPlaceholderText("Enter initial balance")
        self.wallet_input.setStyleSheet("padding: 5px; border: 1px solid
#ccc;")
```

```
        self.create_wallet_btn = QPushButton("Create Wallet")
        self.create_wallet_btn.setStyleSheet("padding: 5px 10px;
background-color: #4CAF50; color: white; border: none; border-radius:
5px;")
        self.create_wallet_btn.clicked.connect(self.create_wallet)

        # Wallet Display
        self.balance_label = QLabel("Balance: $0")
        self.balance_label.setFont(QFont("Arial", 14))
        self.balance_label.setStyleSheet("color: #2c3e50;")

        # Bid Section
        self.item_label = QLabel("Current Item: Rare Painting")
        self.item_label.setFont(QFont("Arial", 12, QFont.Bold))

        self.bid_input = QLineEdit()
        self.bid_input.setPlaceholderText("Enter your bid amount")
        self.bid_input.setStyleSheet("padding: 5px; border: 1px solid
#ccc;")

        self.bid_btn = QPushButton("Place Bid")
        self.bid_btn.setStyleSheet("padding: 5px 10px; background-color:
#2196F3; color: white; border: none; border-radius: 5px;")
        self.bid_btn.clicked.connect(self.place_bid)

        header_layout = QHBoxLayout()
        header_layout.addWidget(self.back_btn, alignment=Qt.AlignLeft)
        header_layout.addStretch()
```

```
        # Layout
        wallet_layout = QHBoxLayout()
        wallet_layout.addWidget(self.wallet_label)
        wallet_layout.addWidget(self.wallet_input)
        wallet_layout.addWidget(self.create_wallet_btn)

        bid_layout = QHBoxLayout()
        bid_layout.addWidget(self.bid_input)
        bid_layout.addWidget(self.bid_btn)

        layout = QVBoxLayout()
        layout.addLayout(header_layout)
        layout.addLayout(wallet_layout)
        layout.addWidget(self.balance_label, alignment=Qt.AlignCenter)
        layout.addWidget(self.item_label, alignment=Qt.AlignCenter)
        layout.addLayout(bid_layout)

        self.setLayout(layout)

    def create_wallet(self):
        try:
            balance = float(self.wallet_input.text())
            if balance < 0:
                raise ValueError("Balance cannot be negative.")
            self.wallet_balance = balance
            self.balance_label.setText(f"Balance:
${self.wallet_balance:.2f}")
            QMessageBox.information(self, "Wallet Created", "Your wallet
has been created successfully!")
```

```
        except ValueError:
            QMessageBox.warning(self, "Invalid Input", "Please enter a
valid positive number for the balance.")


    def place_bid(self):
        try:
            bid_amount = float(self.bid_input.text())
            if bid_amount <= 0:
                raise ValueError("Bid amount must be greater than zero.")


            if bid_amount > self.wallet_balance:
                QMessageBox.warning(self, "Insufficient Balance", "You do
not have enough balance to place this bid.")
            else:
                self.wallet_balance -= bid_amount
                self.balance_label.setText(f"Balance:
${self.wallet_balance:.2f}")
                QMessageBox.information(self, "Bid Successful", f"You have
placed a bid of ${bid_amount:.2f}.")
        except ValueError:
            QMessageBox.warning(self, "Invalid Input", "Please enter a
valid bid amount.")


    def go_back(self):
        try:
            self.destroy()
            obj=main_ui()
            obj.show()
        except Exception as e:
            traceback.print_exc()
```

```
class profile_ui(QDialog):
    def __init__(self):
        super(profile_ui, self).__init__()
        self.setWindowTitle("Settings")
        self.setGeometry(275, 140, 900, 500)  # Increase width but keep
the height moderate

        # MySQL Database connection
        self.conn = sql.connect(
            host="localhost",  # Replace with your MySQL server details
            user="root",  # Replace with your MySQL username
            password="sairam",  # Replace with your MySQL password
            database="aucsof_ip"  # Replace with your database name
        )
        self.cursor = self.conn.cursor()

        # Create the table if it doesn't exist (MySQL syntax)
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS users (
                id INT AUTO_INCREMENT PRIMARY KEY,
                full_name VARCHAR(255),
                email VARCHAR(255),
                phone VARCHAR(20),
                address VARCHAR(255),
                dob DATE,
                bio TEXT,
                linkedin VARCHAR(255),
                password VARCHAR(255)
```

```
            )
        ''')

        # Example: Assuming we're editing the user with id 1 (modify this
as per your logic)
        self.user_id = 1

        # Font settings for UI elements
        self.font = QFont("Garamond", 12)
        self.heading_font = QFont("Garamond", 22, QFont.Bold)

        # Background color for header
        self.tabsbg_LBL = QLabel(self)
        self.tabsbg_LBL.setStyleSheet("background-
color:rgba(0,12,95,205)")
        self.tabsbg_LBL.setGeometry(0, 0, 891, 80)

        # Font settings for labels and input fields
        self.LBL_font = QFont("Garamond", 14, QFont.Bold)

        # Scroll Area for the form
        self.scrollArea = QScrollArea(self)
        self.scrollArea.setObjectName(u"scrollArea")
        self.scrollArea.setGeometry(QRect(20, 120, 861, 350))  # Increased
width of scroll area
        self.scrollArea.setWidgetResizable(True)

        self.scrollAreaWidgetContents = QWidget()
self.scrollAreaWidgetContents.setObjectName(u"scrollAreaWidgetContents")
```

```
        self.scrollAreaWidgetContents.setGeometry(QRect(0, 0, 860, 340))


        self.scrollArea.setWidget(self.scrollAreaWidgetContents)


        # Profile header label
        self.profileheading_LBL = QLabel(self)
        self.profileheading_LBL.setText("Profile Settings")
        self.profileheading_LBL.setGeometry(350, 20, 200, 40)
        self.profileheading_LBL.setFont(self.heading_font)
        self.profileheading_LBL.setStyleSheet("background-color: rgba(255,
255, 255, 10); color : white")


        # Back Button
        self.back_BTN = QPushButton(self)
        self.back_BTN.setText("Back")
        self.back_BTN.setFont(self.LBL_font)
        self.back_BTN.setGeometry(800, 0, 100, 50)
        self.back_BTN.clicked.connect(self.going_back)
        self.back_BTN.setStyleSheet("background-color: rgba(255, 255, 255,
10); color : white")


        # Create the main layout for the scroll area
        self.main_layout = QVBoxLayout(self.scrollAreaWidgetContents)
        self.main_layout.setSpacing(20)


        # Full Name Input
        self.fullname_input = QLineEdit(self)
        self.fullname_input.setFont(self.font)
        self.fullname_input.setPlaceholderText("Enter your full name")
```

```
self.main_layout.addWidget(QLabel("Full Name", self))
self.main_layout.addWidget(self.fullname_input)

# Email Input
self.email_input = QLineEdit(self)
self.email_input.setFont(self.font)
self.email_input.setPlaceholderText("Enter your email")
self.main_layout.addWidget(QLabel("Email Address", self))
self.main_layout.addWidget(self.email_input)

# Phone Number Input
self.phone_input = QLineEdit(self)
self.phone_input.setFont(self.font)
self.phone_input.setPlaceholderText("Enter your phone number")
self.main_layout.addWidget(QLabel("Phone Number", self))
self.main_layout.addWidget(self.phone_input)

# Address Input
self.address_input = QLineEdit(self)
self.address_input.setFont(self.font)
self.address_input.setPlaceholderText("Enter your address")
self.main_layout.addWidget(QLabel("Address", self))
self.main_layout.addWidget(self.address_input)

# Date of Birth Input
self.dob_input = QDateEdit(QDate.currentDate(),self)
self.dob_input.setFont(self.font)
self.dob_input.setCalendarPopup(True)
```

```
        self.dob_input.setDisplayFormat("yyyy-MM-dd")
        self.main_layout.addWidget(QLabel("Date of Birth", self))
        self.main_layout.addWidget(self.dob_input)

        # Bio Input
        self.bio_input = QLineEdit(self)
        self.bio_input.setFont(self.font)
        self.bio_input.setPlaceholderText("Tell us something about you")
        self.main_layout.addWidget(QLabel("Bio", self))
        self.main_layout.addWidget(self.bio_input)

        # Password Section
        self.change_pass_label = QLabel("Change Password", self)
        self.change_pass_label.setFont(self.font)
        self.main_layout.addWidget(self.change_pass_label)

        self.old_password_input = QLineEdit(self)
        self.old_password_input.setFont(self.font)
        self.old_password_input.setPlaceholderText("Enter old password")
        self.old_password_input.setEchoMode(QLineEdit.Password)
        self.main_layout.addWidget(QLabel("Old Password", self))
        self.main_layout.addWidget(self.old_password_input)

        self.new_password_input = QLineEdit(self)
        self.new_password_input.setFont(self.font)
        self.new_password_input.setPlaceholderText("Enter new password")
        self.new_password_input.setEchoMode(QLineEdit.Password)
        self.main_layout.addWidget(QLabel("New Password", self))
```

```
        self.main_layout.addWidget(self.new_password_input)


        self.confirm_password_input = QLineEdit(self)
        self.confirm_password_input.setFont(self.font)
        self.confirm_password_input.setPlaceholderText("Confirm new
password")
        self.confirm_password_input.setEchoMode(QLineEdit.Password)
        self.main_layout.addWidget(QLabel("Confirm Password", self))
        self.main_layout.addWidget(self.confirm_password_input)


        # Create a layout for buttons
        self.button_layout = QHBoxLayout()


        # Save Changes Button
        self.save_btn = QPushButton("Save Changes", self)
        self.save_btn.setFont(self.LBL_font)
        self.save_btn.setStyleSheet("background-color: #4CAF50; color:
white")
        self.save_btn.clicked.connect(self.save_changes)
        self.button_layout.addWidget(self.save_btn)


        # Cancel Button
        self.cancel_btn = QPushButton("Cancel", self)
        self.cancel_btn.setFont(self.LBL_font)
        self.cancel_btn.setStyleSheet("background-color: #f44336; color:
white")
        self.cancel_btn.clicked.connect(self.cancel_changes)
        self.button_layout.addWidget(self.cancel_btn)
```

```
        # Add button layout to the main layout
        self.main_layout.addLayout(self.button_layout)

        # Load the current user data
        self.load_user_data()

    def going_back(self):
        # Logic to go back to the previous screen
        self.destroy()
        self.main_uiobj = main_ui()
        self.main_uiobj.show()

    def load_user_data(self):
        # Fetch user data from MySQL database (assuming the user ID is 1
for now)
        self.cursor.execute("SELECT * FROM users WHERE id=%s",
(self.user_id,))
        user = self.cursor.fetchone()

        if user:
            self.fullname_input.setText(user[1])
            self.email_input.setText(user[2])
            self.phone_input.setText(user[3])
            self.address_input.setText(user[4])
            self.dob_input.setText(user[5])
            self.bio_input.setText(user[6])

    def save_changes(self):
        # Logic to save changes to MySQL database
```

```
        fullname = self.fullname_input.text()
        email = self.email_input.text()
        phone = self.phone_input.text()
        address = self.address_input.text()
        dob = self.dob_input.text()
        bio = self.bio_input.text()
        old_password = self.old_password_input.text()
        new_password = self.new_password_input.text()
        confirm_password = self.confirm_password_input.text()

        # Simple password change check
        if new_password != confirm_password:
            QMessageBox.warning(self, "Error", "Passwords do not match!")
            return

        # Hash the password or store it securely (add hashing logic here
if needed)
        if new_password:  # Only update password if it's provided
            password = new_password
        else:
            password = old_password  # Or fetch the current password if
unchanged

        self.cursor.execute("""
            UPDATE users SET
            full_name=%s,
            email=%s,
            phone=%s,
            address=%s,
```

```
            dob=%s,
            bio=%s,
            password=%s
            WHERE id=%s
        """, (fullname, email, phone, address, dob, bio, password,
self.user_id))


        self.conn.commit()
        QMessageBox.information(self, "Success", "Profile updated
successfully!")


    def cancel_changes(self):
        self.load_user_data()


if __name__=="__main__":
    app = QApplication(sys.argv)
    l = login()
    l.show()
    app.exec_()
```

Future Enhancements

- Enable the platform to host multiple auctions at the same time, allowing users to participate in various auctions simultaneously.
- Introduce different user roles (e.g., admin, bidder, seller) with specific permissions to manage access and functionalities effectively.
- Support various auction formats such as Dutch, sealed-bid, and reverse auctions, catering to diverse user needs and preferences.
- Implement real-time notifications for bid changes, ensuring users receive instant updates without needing to refresh the page.
- Allow users to set a maximum bid amount that automatically raises their bid if they are outbid, streamlining the bidding process.
- Develop a mobile application to provide users with easy access to auctions and bidding from their smartphones.
- Integrate multiple payment gateways (e.g., credit cards, PayPal) for seamless transactions during and after auctions.
- Provide access to detailed auction histories, including past bids and winners, allowing users to analyze trends and make informed decisions.
- Enable users to rate and review sellers post-auction to build trust and assist future bidders in making informed choices.
- Introduce a verification process for high-value items to confirm authenticity, enhancing buyer confidence.

System Requirements

For a smooth, fast and user-friendly experience, the following system requirements are recommended for the client user:

- A system with 8 GB RAM.
- An Intel(R) Core(TM) i3-6100 CPU @ 3.70GHz processor.
- A minimum screen size of 1366×768.

Python 3.11 which was used in the making of this project functions well with:

- Windows 7,10 or 11
- Mac OS X 10.11 or higher, 64-bit
- Linux: RHEL 6/7, 64-bit (almost all libraries also work in Ubuntu)
- 64-bit x86 CPU (Intel / AMD architecture). ARM CPUs are not supported.
- 4 GB RAM and 5 GB free disk space.

MySQL 8.0 which was used in the making of this project has the following requirements:

Minimum Hardware Requirements

- 2 GB RAM
- 2 CPU Cores
- Disk I/O subsystem applicable to a write-intensive database Recommended Hardware

REQUIREMENTS

- 4 CPU Cores or more
- 8 GB RAM or more

Bibiliography

- CBSE Informatics Practices Textbook
- www.geeksforgeeks.com
- www.w3schools.com
- www.github.com