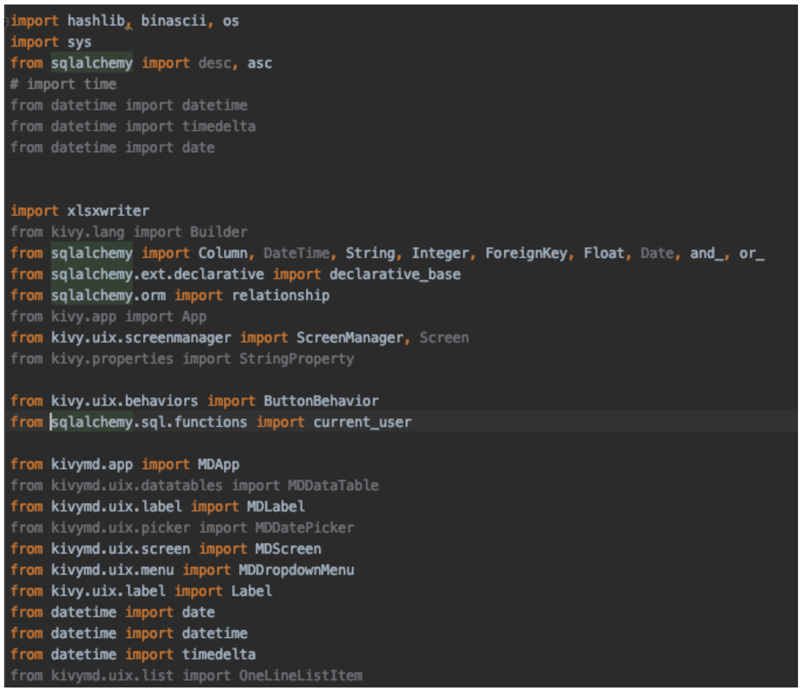
Criterion C: Development

Technique used
-Variables
-Loops
-Bubble sort
-List data structure
-Nested conditionals
-User-defined functions
-Database management (Sqlite3)
-Desktop application framework (Kivy)
-Various libraries (ex. datetime)

Tools used
-Import Kivy and kivymd libraries into the program

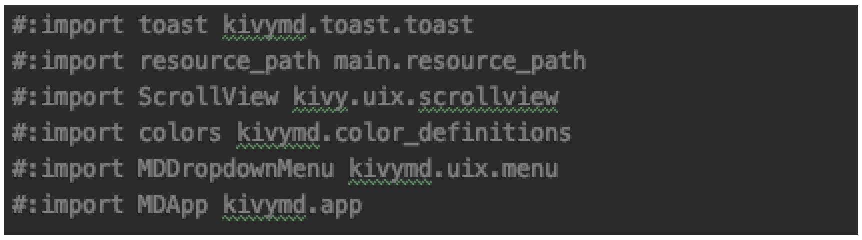
Configuration

To start developing the program, I need to configure its base first.
I created 2 main files: main.py (a Python file) for back end coding, and main.kv (a Kivy file) for front end coding. I then imported various libraries including KivyMD in both of these files in order to provide more functions for my program (ex. I imported the library date so I can get the date the day the client adds an invoice) as follow:



```
import hashlib, binascii, os
import sys
from sqlalchemy import desc, asc
# import time
from datetime import datetime
from datetime import timedelta
from datetime import date


import xlsxwriter
from kivy.lang import Builder
from sqlalchemy import Column, DateTime, String, Integer, ForeignKey, Float, Date, and_, or_
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship
from kivy.app import App
from kivy.uix.screenmanager import ScreenManager, Screen
from kivy.properties import StringProperty

from kivy.uix.behaviors import ButtonBehavior
from sqlalchemy.sql.functions import current_user

from kivymd.app import MDApp
from kivymd.uix.datatables import MDDataTable
from kivymd.uix.label import MDLabel
from kivymd.uix.picker import MDDatePicker
from kivymd.uix.screen import MDScreen
from kivymd.uix.menu import MDDropdownMenu
from kivy.uix.label import Label
from datetime import date
from datetime import datetime
from datetime import timedelta
from kivymd.uix.list import OneLineListItem
```

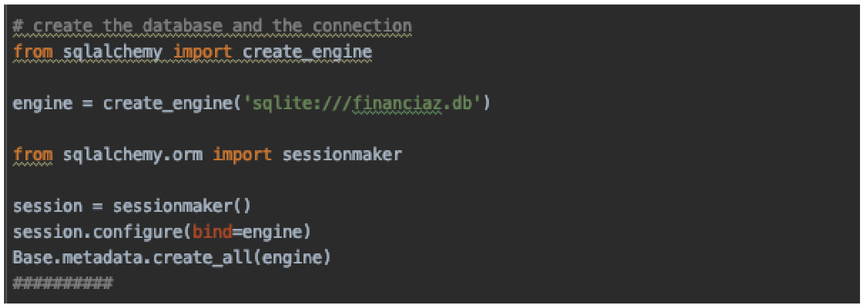
Figure 1: Importing libraries (main.py)



```
#:import toast kivymd.toast.toast
#:import resource_path main.resource_path
#:import ScrollView kivy.uix.scrollview
#:import colors kivymd.color_definitions
#:import MDDropdownMenu kivymd.uix.menu
#:import MDApp kivymd.app
```

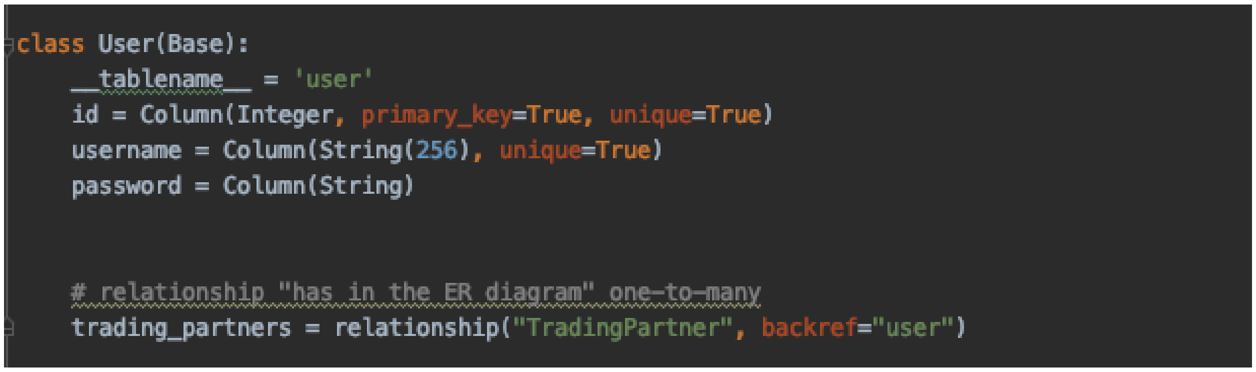
Figure 2: Importing libraries (main.kv)

I then created the 3 data tables (User, Trading partner, Invoice) in main.py:



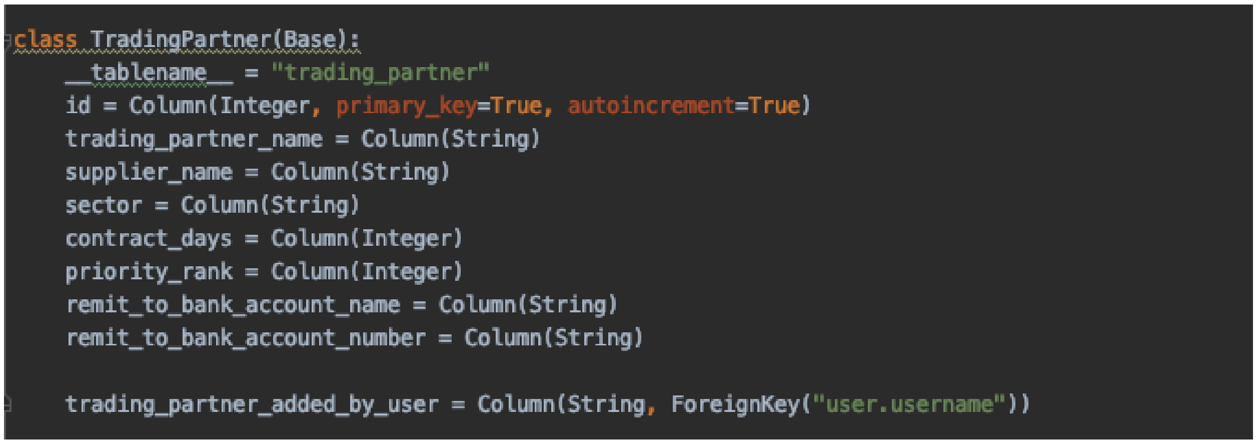
```
# create the database and the connection
from sqlalchemy import create_engine

engine = create_engine('sqlite:///financiaz.db')

from sqlalchemy.orm import sessionmaker

session = sessionmaker()
session.configure(bind=engine)
Base.metadata.create_all(engine)
##########
```

Figure 3: Creating the sqlite database for my program (main.py)



```python
class User(Base):
    __tablename__ = 'user'
    id = Column(Integer, primary_key=True, unique=True)
    username = Column(String(256), unique=True)
    password = Column(String)


    # relationship "has in the ER diagram" one-to-many
    trading_partners = relationship("TradingPartner", backref="user")
```

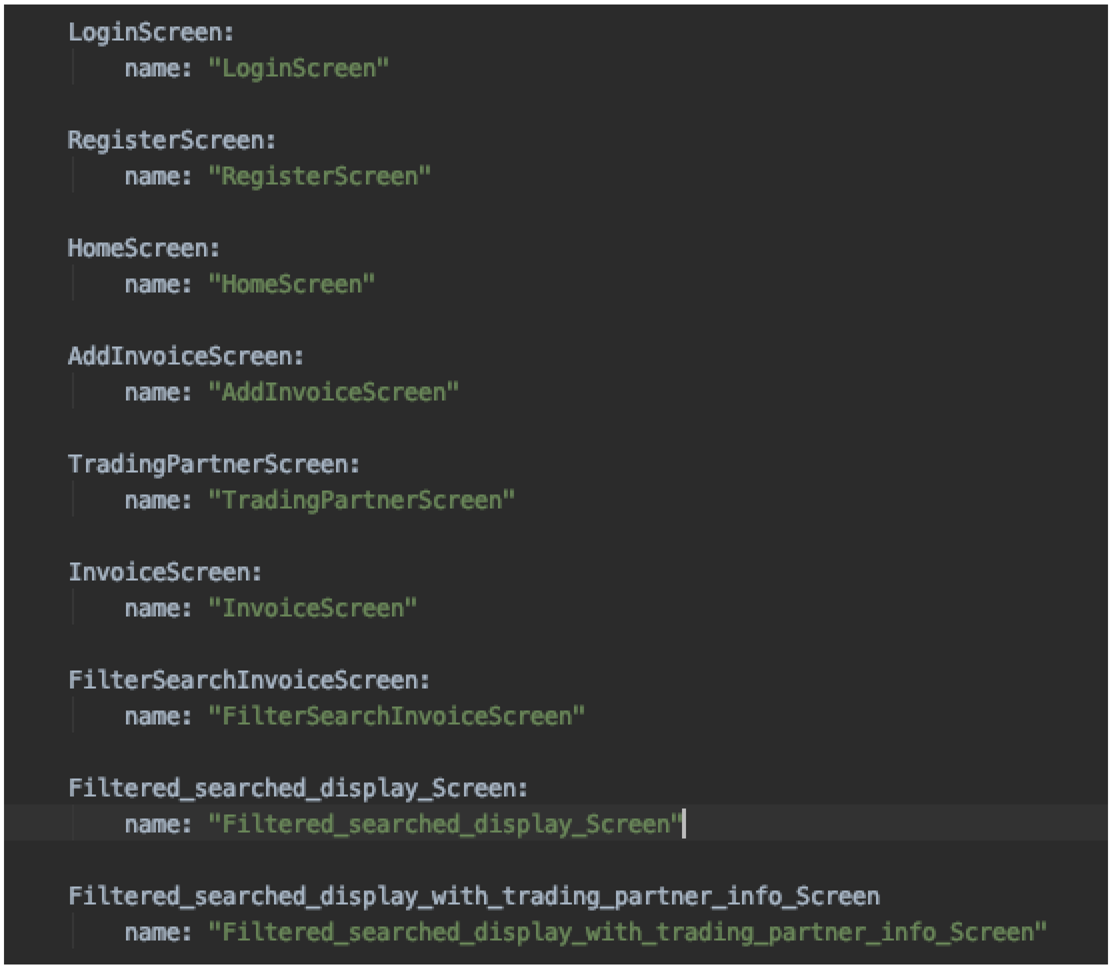
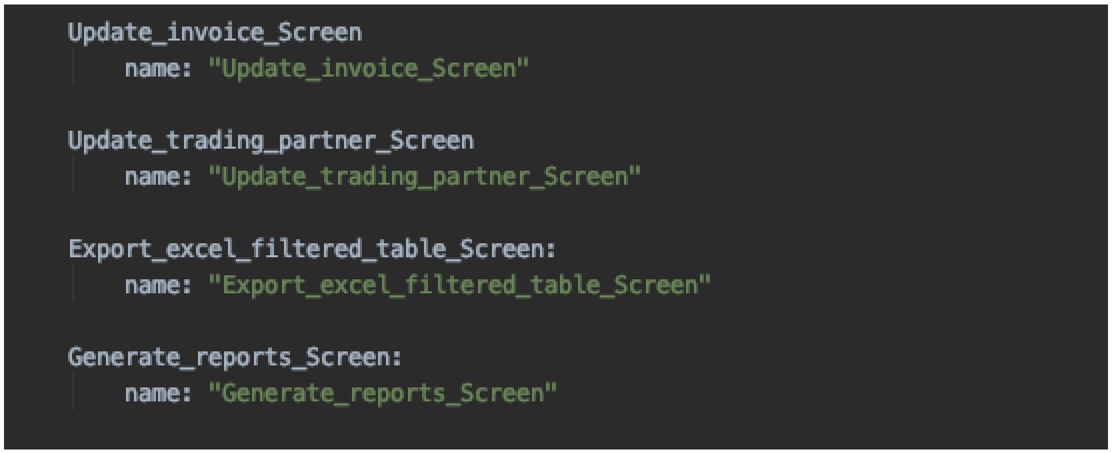
Figure 4: Creating the User data table (main.py)



```python
class TradingPartner(Base):
    __tablename__ = "trading_partner"
    id = Column(Integer, primary_key=True, autoincrement=True)
    trading_partner_name = Column(String)
    supplier_name = Column(String)
    sector = Column(String)
    contract_days = Column(Integer)
    priority_rank = Column(Integer)
    remit_to_bank_account_name = Column(String)
    remit_to_bank_account_number = Column(String)

    trading_partner_added_by_user = Column(String, ForeignKey("user.username"))
```

Figure 5: Creating the Trading partner table (main.py)

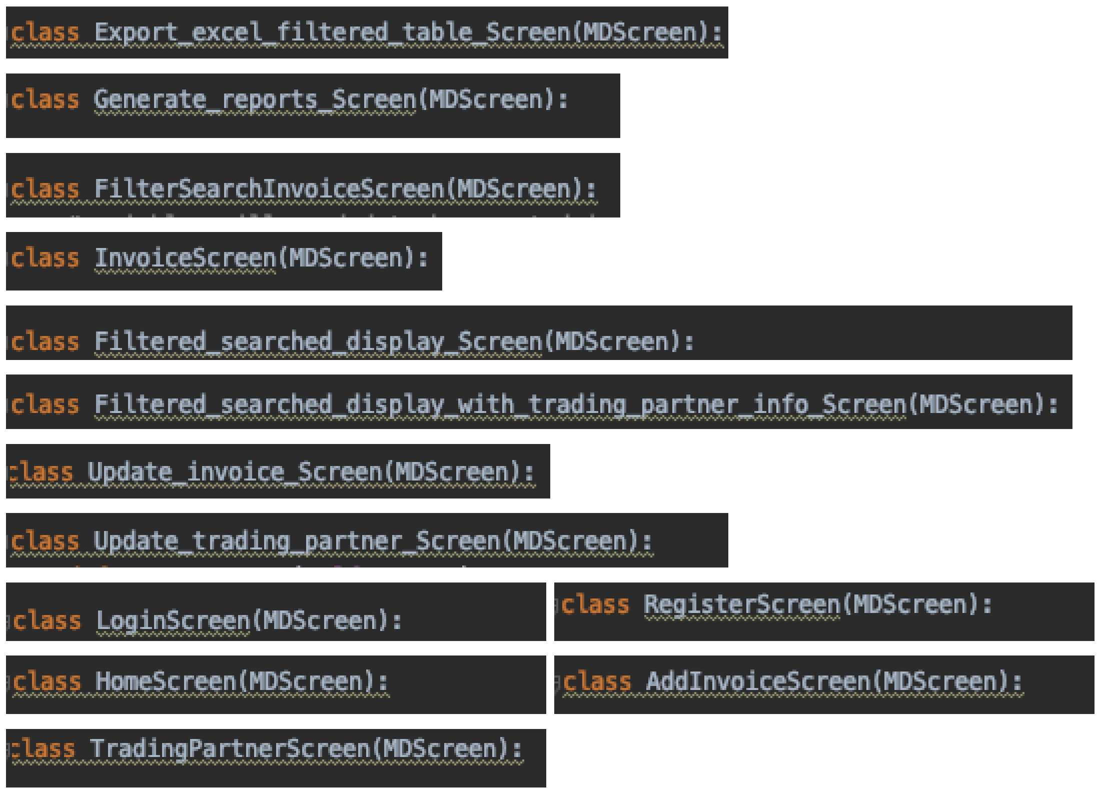
Then, I created front-end screens on main.kv and matched each screen to a class on main.py:

```
LoginScreen:
    name: "LoginScreen"

RegisterScreen:
    name: "RegisterScreen"

HomeScreen:
    name: "HomeScreen"

AddInvoiceScreen:
    name: "AddInvoiceScreen"

TradingPartnerScreen:
    name: "TradingPartnerScreen"

InvoiceScreen:
    name: "InvoiceScreen"

FilterSearchInvoiceScreen:
    name: "FilterSearchInvoiceScreen"

Filtered_searched_display_Screen:
    name: "Filtered_searched_display_Screen"

Filtered_searched_display_with_trading_partner_info_Screen
    name: "Filtered_searched_display_with_trading_partner_info_Screen"
```

```
Update_invoice_Screen
    name: "Update_invoice_Screen"

Update_trading_partner_Screen
    name: "Update_trading_partner_Screen"

Export_excel_filtered_table_Screen:
    name: "Export_excel_filtered_table_Screen"

Generate_reports_Screen:
    name: "Generate_reports_Screen"
```

Figure 6: Creating front end screens on main.kv



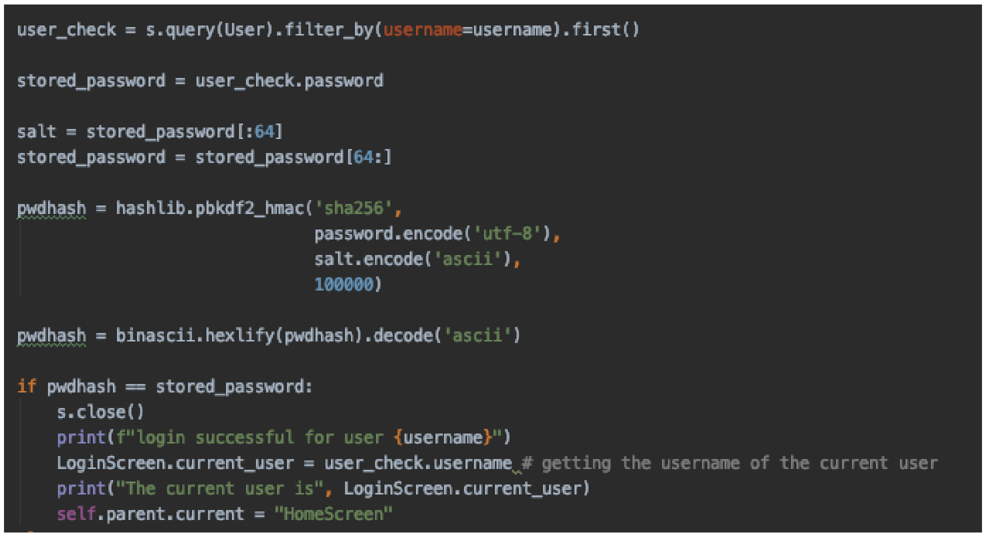
```
class Export_excel_filtered_table_Screen(MDScreen):
class Generate_reports_Screen(MDScreen):
class FilterSearchInvoiceScreen(MDScreen):
class InvoiceScreen(MDScreen):
class Filtered_searched_display_Screen(MDScreen):
class Filtered_searched_display_with_trading_partner_info_Screen(MDScreen):
class Update_invoice_Screen(MDScreen):
class Update_trading_partner_Screen(MDScreen):
class LoginScreen(MDScreen):
class RegisterScreen(MDScreen):
class HomeScreen(MDScreen):
class AddInvoiceScreen(MDScreen):
class TradingPartnerScreen(MDScreen):
```

Figure 7: Creating each class on main.py for each screen on main.kv

<u>Development</u>

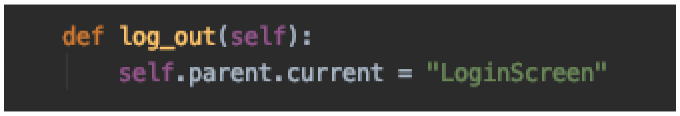
1.The program will have a login and logout screen
I had shown the way I created every screen for my program above. However, I would like to explain the codes I use to make these 2 screens have the login and logout functions.

On the Login screen, I first query the user (the client or his employee) from the database and check if the hash value of their input password is the same as the hash password stored in the database. If yes, I will let the user into the HomeScreen of the program, where they will be able to click on buttons that provide them the functions of the program. The HomeScreen also acts as the logout screen, having a logout button that,when the user clicks it, lets the user back to the Login screen.



```
user_check = s.query(User).filter_by(username=username).first()

stored_password = user_check.password

salt = stored_password[:64]
stored_password = stored_password[64:]

pwdhash = hashlib.pbkdf2_hmac('sha256',
                              password.encode('utf-8'),
                              salt.encode('ascii'),
                              100000)

pwdhash = binascii.hexlify(pwdhash).decode('ascii')

if pwdhash == stored_password:
    s.close()
    print(f"login successful for user {username}")
    LoginScreen.current_user = user_check.username # getting the username of the current user
    print("The current user is", LoginScreen.current_user)
    self.parent.current = "HomeScreen"
```

Figure 8: Login validation algorithm (main.py)

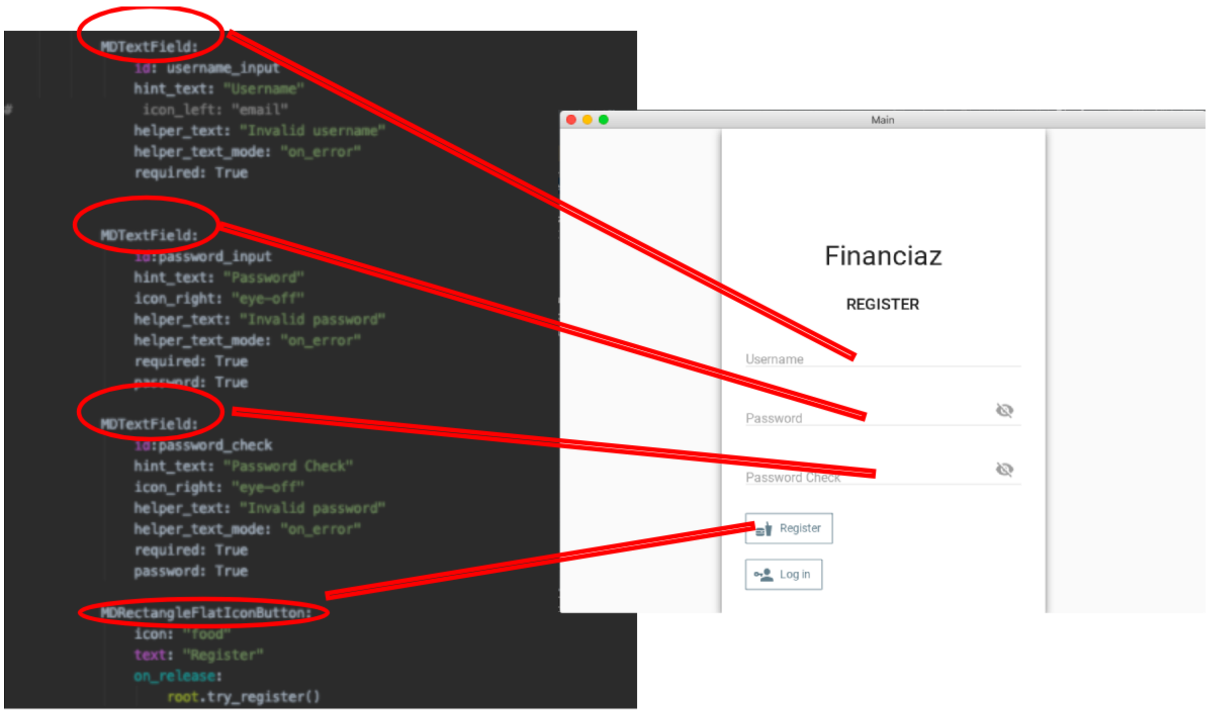


```
    def log_out(self):
        self.parent.current = "LoginScreen"
```

Figure 9: Logout algorithm (main.py)

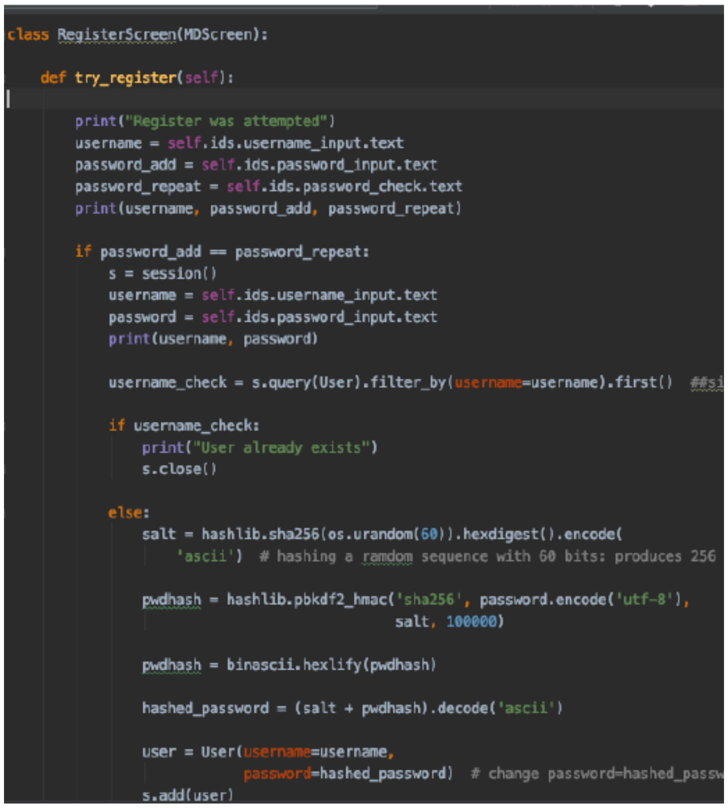
2.The program will let multiple user creates their account and passwords, storing those passwords in hashed

New users can create their username and passwords and input them into the program (through the 2 MDTextField on the RegisterScreen in Figure 10). As they register, clicking the “Register” button (Figure 10), they will call the function “def try_register ()” in the python file in Figure 11. The program will query to check if the registered username already existed by doing an SQLAlchemy query. If it doesn't exist, the program will register the user into the database and hash the registered password with the library “hashlib” and “sha256”.



```
MDTextField:
    id: username_input
    hint_text: "Username"
     icon_left: "email"
    helper_text: "Invalid username"
    helper_text_mode: "on_error"
    required: True

MDTextField:
    id:password_input
    hint_text: "Password"
    icon_right: "eye-off"
    helper_text: "Invalid password"
    helper_text_mode: "on_error"
    required: True
    password: True

MDTextField:
    id:password_check
    hint_text: "Password Check"
    icon_right: "eye-off"
    helper_text: "Invalid password"
    helper_text_mode: "on_error"
    required: True
    password: True

MDRectangleFlatIconButton:
    icon: "food"
    text: "Register"
    on_release:
        root.try_register()
```

Figure 10: Showing the RegisterScreen's GUI and its backend Kivy components in main.kv

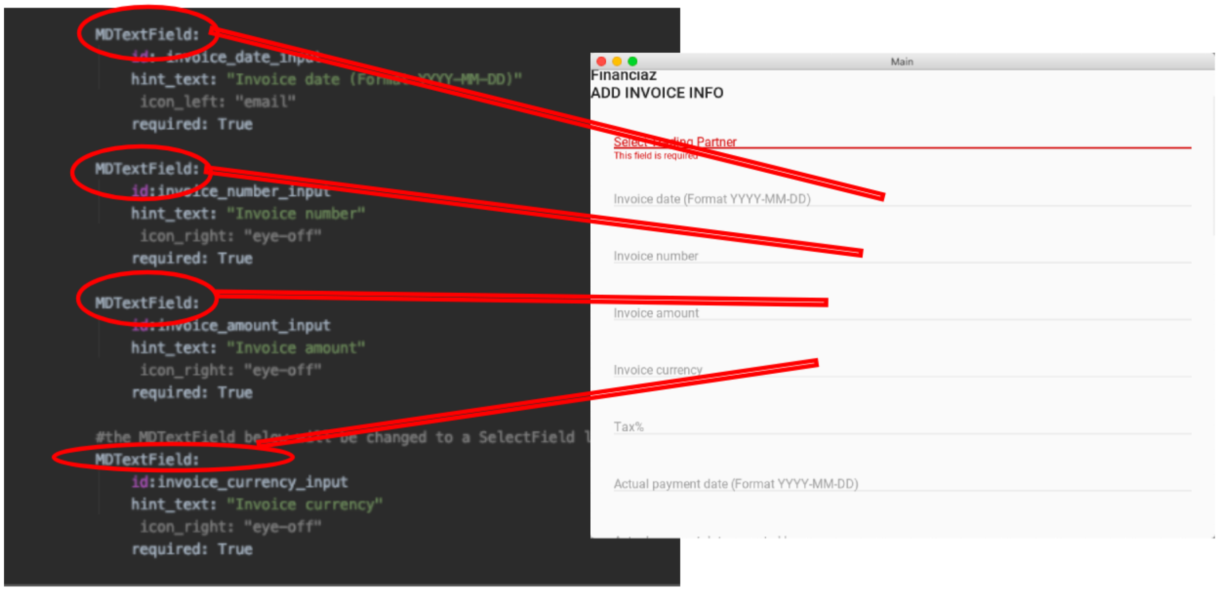


```python
class RegisterScreen(MDScreen):

    def try_register(self):

        print("Register was attempted")
        username = self.ids.username_input.text
        password_add = self.ids.password_input.text
        password_repeat = self.ids.password_check.text
        print(username, password_add, password_repeat)

        if password_add == password_repeat:
            s = session()
            username = self.ids.username_input.text
            password = self.ids.password_input.text
            print(username, password)

            username_check = s.query(User).filter_by(username=username).first()  ##si

            if username_check:
                print("User already exists")
                s.close()

            else:
                salt = hashlib.sha256(os.urandom(60)).hexdigest().encode(
                    'ascii')  # hashing a ramdom sequence with 60 bits: produces 256

                pwdhash = hashlib.pbkdf2_hmac('sha256', password.encode('utf-8'),
                                              salt, 100000)

                pwdhash = binascii.hexlify(pwdhash)

                hashed_password = (salt + pwdhash).decode('ascii')

                user = User(username=username,
                            password=hashed_password)  # change password=hashed_passw
                s.add(user)
```

Figure 11: Showing the def try_register() function for the RegisterScreen in the back end (main.py)

3.Let the client add/edit/remove invoices.

First, to add an invoice, the user will input info of the invoices through multiple MDTextField on the InvoiceScreen (Figure 12) and click a button. Then the program will call the function “def add_to_database()” and store all the input values from the user temporarily into multiple variables (Figure 13).



```
MDTextField:
    id: invoice_date_input
    hint_text: "Invoice date (Format YYYY-MM-DD)"
     icon_left: "email"
    required: True

MDTextField:
    id:invoice_number_input
    hint_text: "Invoice number"
     icon_right: "eye-off"
    required: True

MDTextField:
    id:invoice_amount_input
    hint_text: "Invoice amount"
     icon_right: "eye-off"
    required: True

#the MDTextField below will be changed to a SelectField l
MDTextField:
    id:invoice_currency_input
    hint_text: "Invoice currency"
     icon_right: "eye-off"
    required: True
```

Main

Financiaz

ADD INVOICE INFO

Select Trading Partner

This field is required

Invoice date (Format YYYY-MM-DD)

Invoice number

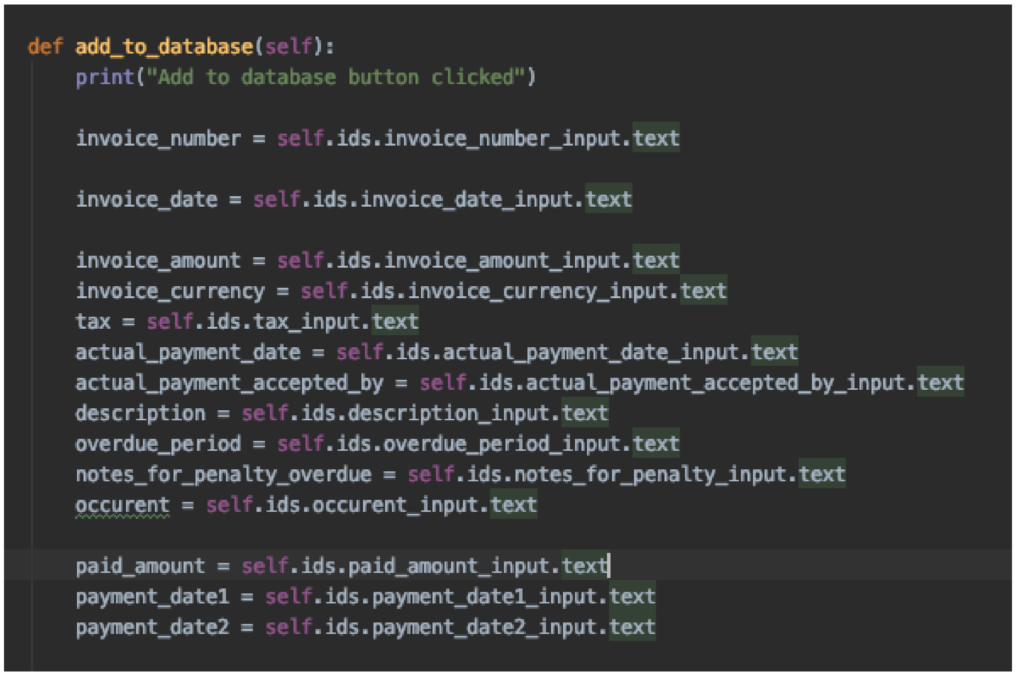
Invoice amount

Invoice currency

Tax%

Actual payment date (Format YYYY-MM-DD)

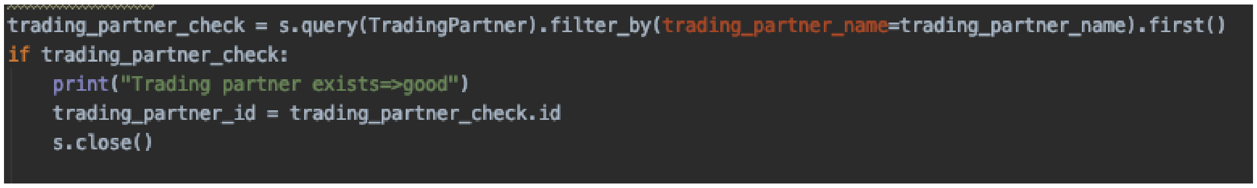
Figure 12: Showing the InvoiceScreen's GUI and its backend Kivy components in main.kv



```
def add_to_database(self):
    print("Add to database button clicked")

    invoice_number = self.ids.invoice_number_input.text

    invoice_date = self.ids.invoice_date_input.text

    invoice_amount = self.ids.invoice_amount_input.text
    invoice_currency = self.ids.invoice_currency_input.text
    tax = self.ids.tax_input.text
    actual_payment_date = self.ids.actual_payment_date_input.text
    actual_payment_accepted_by = self.ids.actual_payment_accepted_by_input.text
    description = self.ids.description_input.text
    overdue_period = self.ids.overdue_period_input.text
    notes_for_penalty_overdue = self.ids.notes_for_penalty_input.text
    occurent = self.ids.occurent_input.text

    paid_amount = self.ids.paid_amount_input.text
    payment_date1 = self.ids.payment_date1_input.text
    payment_date2 = self.ids.payment_date2_input.text
```

Figure 13: Showing the def add_to_database() function for the InvoiceScreen in the back end (main.py)

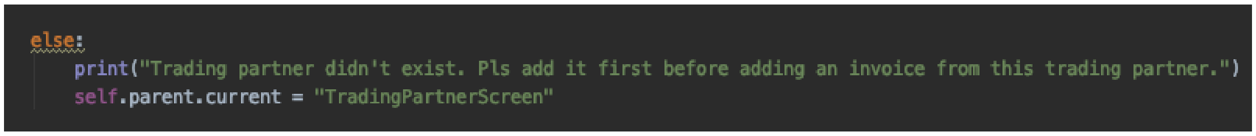
As every invoice requires a registered trading partner, the program will check if the input trading partner already exists or not:



```
trading_partner_check = s.query(TradingPartner).filter_by(trading_partner_name=trading_partner_name).first()
if trading_partner_check:
    print("Trading partner exists=>good")
    trading_partner_id = trading_partner_check.id
    s.close()
```

Figure 14: Showing the trading partner checking algorithm no.1 (main.py)

If the trading partner exists, the program continues. If not, the program will not add the invoice:

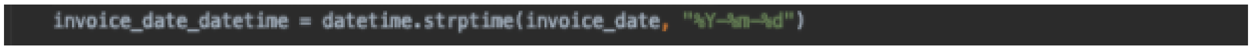


```
else:
    print("Trading partner didn't exist. Pls add it first before adding an invoice from this trading partner.")
    self.parent.current = "TradingPartnerScreen"
```

Figure 15: Showing the trading partner checking algorithm no.2 (main.py)

While the program can add multiple input invoices values into the database for the user, it has to do a few more codes in the backgrounds to create more info for invoices before adding it to the database as following:

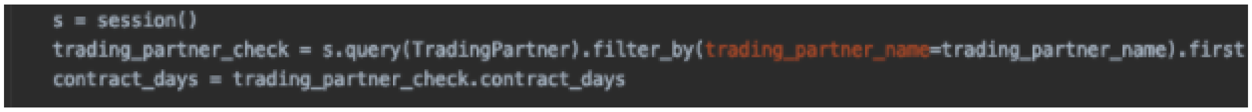
1)



```
    invoice_date_datetime = datetime.strptime(invoice_date, "%Y-%m-%d")
```

Figure 16: Use the current date when the user use the program as the information for the invoice date

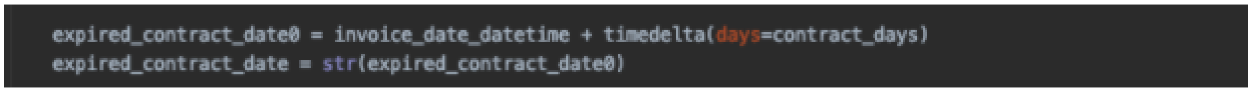
2)



```
    s = session()
    trading_partner_check = s.query(TradingPartner).filter_by(trading_partner_name=trading_partner_name).first
    contract_days = trading_partner_check.contract_days
```

Figure 17: Create the information for the contract days of the input invoice based on the input trading partner of the invoice

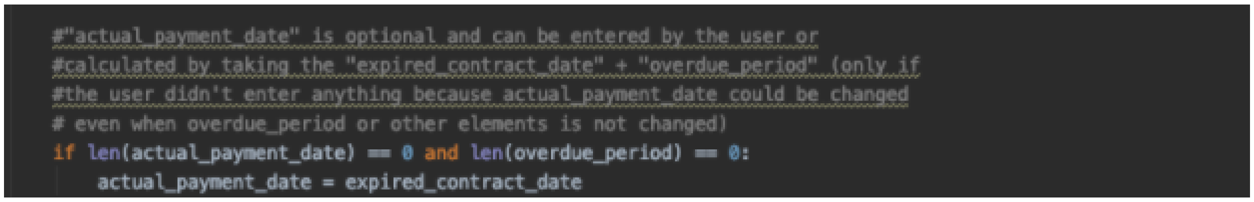
3)



```
expired_contract_date0 = invoice_date_datetime + timedelta(days=contract_days)
expired_contract_date = str(expired_contract_date0)
```

Figure 18: Create the information for the expired contract date based on the just created invoice date and the contract days for the input invoice

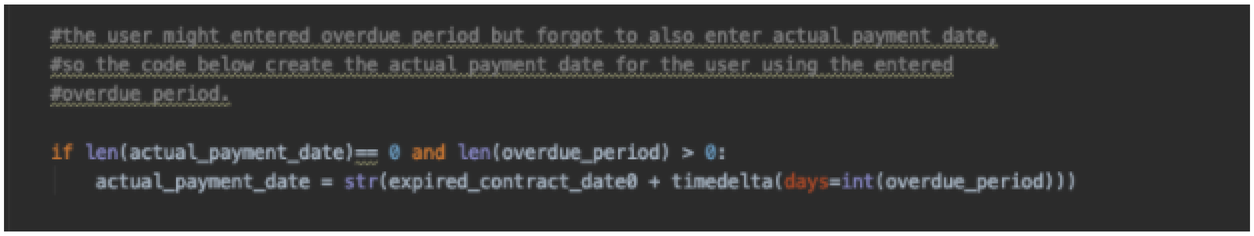
4)



```
#"actual_payment_date" is optional and can be entered by the user or
#calculated by taking the "expired_contract_date" + "overdue_period" (only if
#the user didn't enter anything because actual_payment_date could be changed
# even when overdue_period or other elements is not changed)
if len(actual_payment_date) == 0 and len(overdue_period) == 0:
    actual_payment_date = expired_contract_date
```

Figure 19: Create the actual payment date as the expired contract date of the input invoice if the user didn't input both the actual payment date and the overdue period

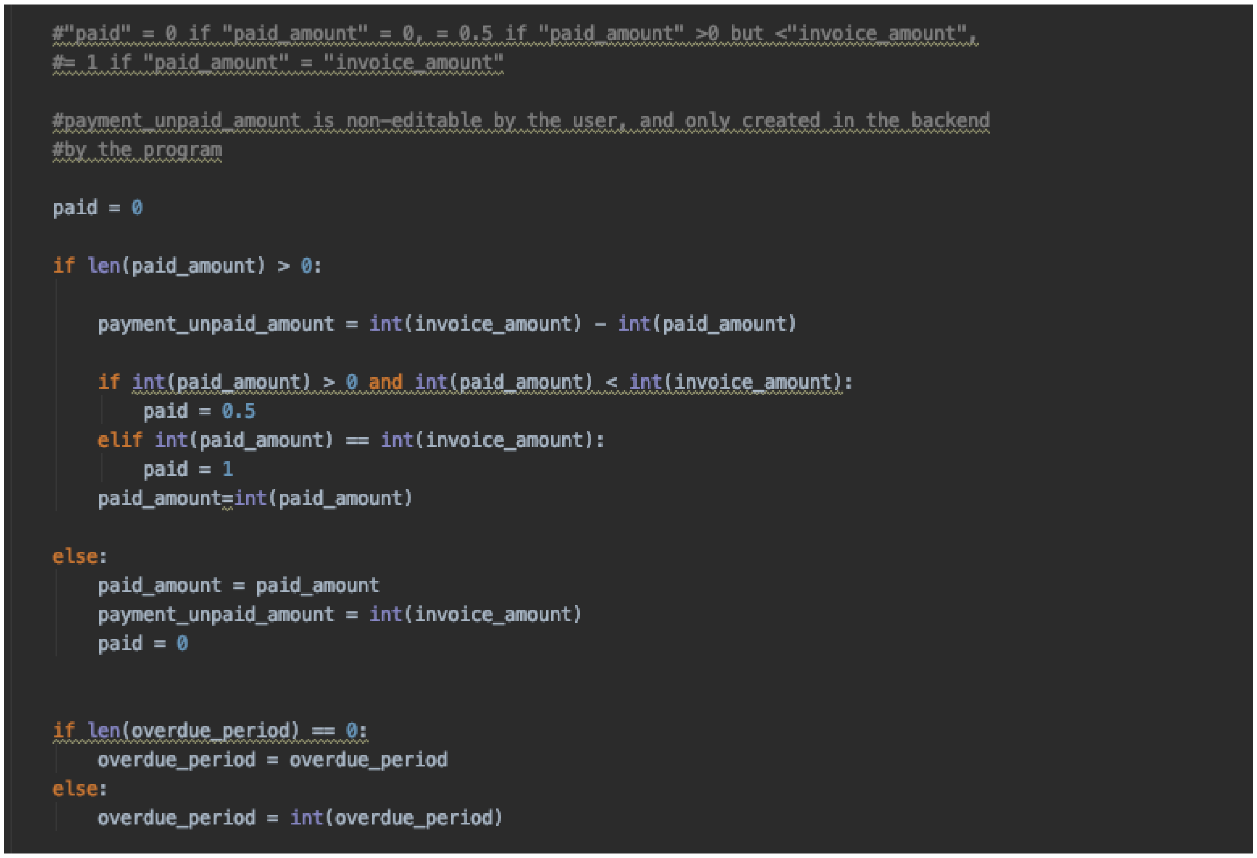
5)



```
#the user might entered overdue period but forgot to also enter actual payment date,
#so the code below create the actual payment date for the user using the entered
#overdue period.

if len(actual_payment_date)== 0 and len(overdue_period) > 0:
    actual_payment_date = str(expired_contract_date0 + timedelta(days=int(overdue_period)))
```

Figure 20: Create the actual payment date as the expired contract date + overdue period of the input invoice if the user didn't input the actual payment date but did input the overdue period

6)

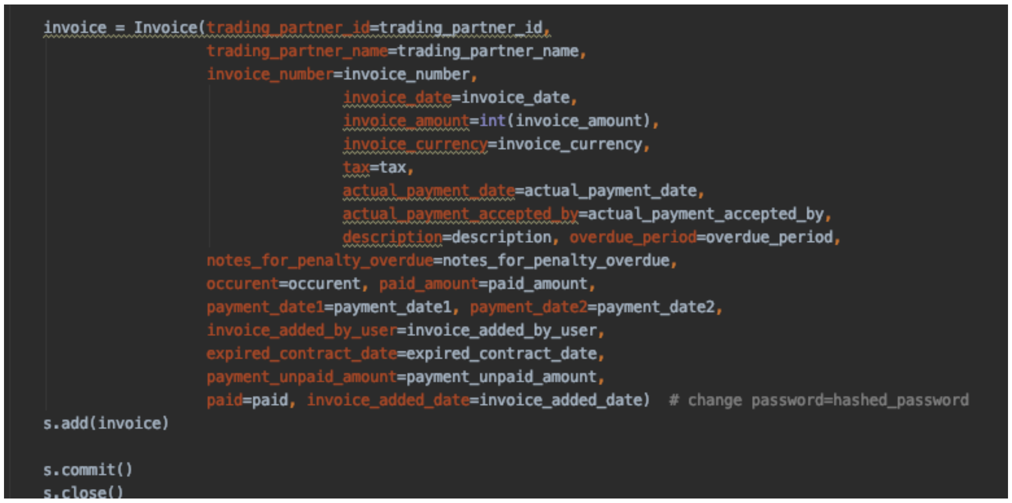


```python
#"paid" = 0 if "paid_amount" = 0, = 0.5 if "paid_amount" >0 but <"invoice_amount",
#= 1 if "paid_amount" = "invoice_amount"

#payment_unpaid_amount is non-editable by the user, and only created in the backend
#by the program

paid = 0

if len(paid_amount) > 0:

    payment_unpaid_amount = int(invoice_amount) - int(paid_amount)

    if int(paid_amount) > 0 and int(paid_amount) < int(invoice_amount):
        paid = 0.5
    elif int(paid_amount) == int(invoice_amount):
        paid = 1
    paid_amount=int(paid_amount)

else:
    paid_amount = paid_amount
    payment_unpaid_amount = int(invoice_amount)
    paid = 0


if len(overdue_period) == 0:
    overdue_period = overdue_period
else:
    overdue_period = int(overdue_period)
```

Figure 21: Create the information for the paid data column of the invoice based by comparing value of the input paid amount and of the invoice amount

The algorithm for the 6 steps is clever because it allows the program to create more important information about the invoice or for the management of the invoices without requiring the users to input them.

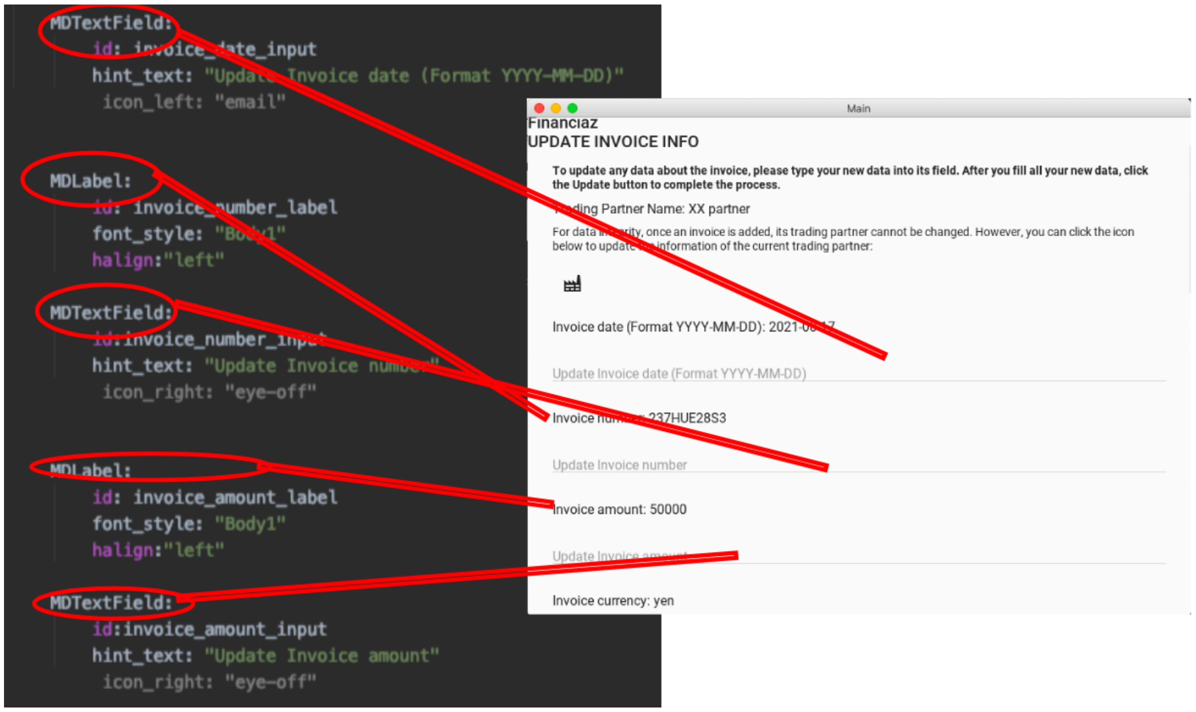
After these 6 steps, the program will add the input invoice from the user into the Invoice data table in the database:



```
invoice = Invoice(trading_partner_id=trading_partner_id,
                  trading_partner_name=trading_partner_name,
                  invoice_number=invoice_number,
                                 invoice_date=invoice_date,
                                 invoice_amount=int(invoice_amount),
                                 invoice_currency=invoice_currency,
                                 tax=tax,
                                 actual_payment_date=actual_payment_date,
                                 actual_payment_accepted_by=actual_payment_accepted_by,
                                 description=description, overdue_period=overdue_period,
                  notes_for_penalty_overdue=notes_for_penalty_overdue,
                  occurent=occurent, paid_amount=paid_amount,
                  payment_date1=payment_date1, payment_date2=payment_date2,
                  invoice_added_by_user=invoice_added_by_user,
                  expired_contract_date=expired_contract_date,
                  payment_unpaid_amount=payment_unpaid_amount,
                  paid=paid, invoice_added_date=invoice_added_date)  # change password=hashed_password
s.add(invoice)

s.commit()
s.close()
```

Figure 22: Algorithm that adds input information alongs with automatically generated information of an input invoice into the database (main.py)

Second, **to edit an invoice,** very similar to adding the invoice, the user will have to input the update information about the invoice through multiple MDTextField on the Update invoice Screen (Figure 23), then click a button, which will call the function “def update_invoice()” in the python file (Figure 24). For most update info, the program will immediately store the new value and update it. (Figure 25). However, for some values that can change each other, the program takes further action similar to when the user input a new invoice. (Figure 16~21)



```
MDTextField:
    id: invoice_date_input
    hint_text: "Update Invoice date (Format YYYY-MM-DD)"
     icon_left: "email"

MDLabel:
    id: invoice_number_label
    font_style: "Body1"
    halign:"left"

MDTextField:
    id:invoice_number_input
    hint_text: "Update Invoice number"
     icon_right: "eye-off"

MDLabel:
    id: invoice_amount_label
    font_style: "Body1"
    halign:"left"

MDTextField:
    id:invoice_amount_input
    hint_text: "Update Invoice amount"
     icon_right: "eye-off"
```

Main

Financiaz

UPDATE INVOICE INFO

To update any data about the invoice, please type your new data into its field. After you fill all your new data, click the Update button to complete the process.

Trading Partner Name: XX partner

For data integrity, once an invoice is added, its trading partner cannot be changed. However, you can click the icon below to update the information of the current trading partner:

Invoice date (Format YYYY-MM-DD): 2021-0[illegible]17

Update Invoice date (Format YYYY-MM-DD)

Invoice number: 237HUE28S3

Update Invoice number

Invoice amount: 50000

Update Invoice amount

Invoice currency: yen

Figure 23: Showing the Update invoice Screen's GUI and its backend Kivy components in main.kv

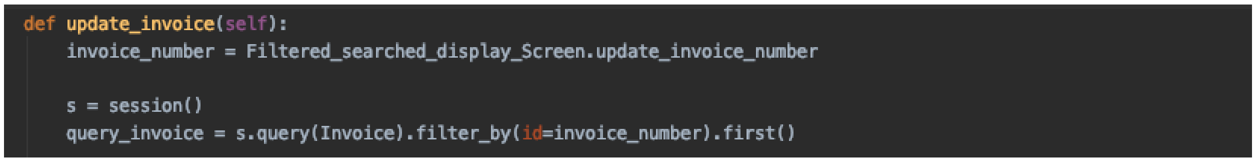

```
def update_invoice(self):
    invoice_number = Filtered_searched_display_Screen.update_invoice_number

    s = session()
    query_invoice = s.query(Invoice).filter_by(id=invoice_number).first()
```

Figure 24: Showing the def update_invoice() function for the Update invoice Screen in the back end (main.py)

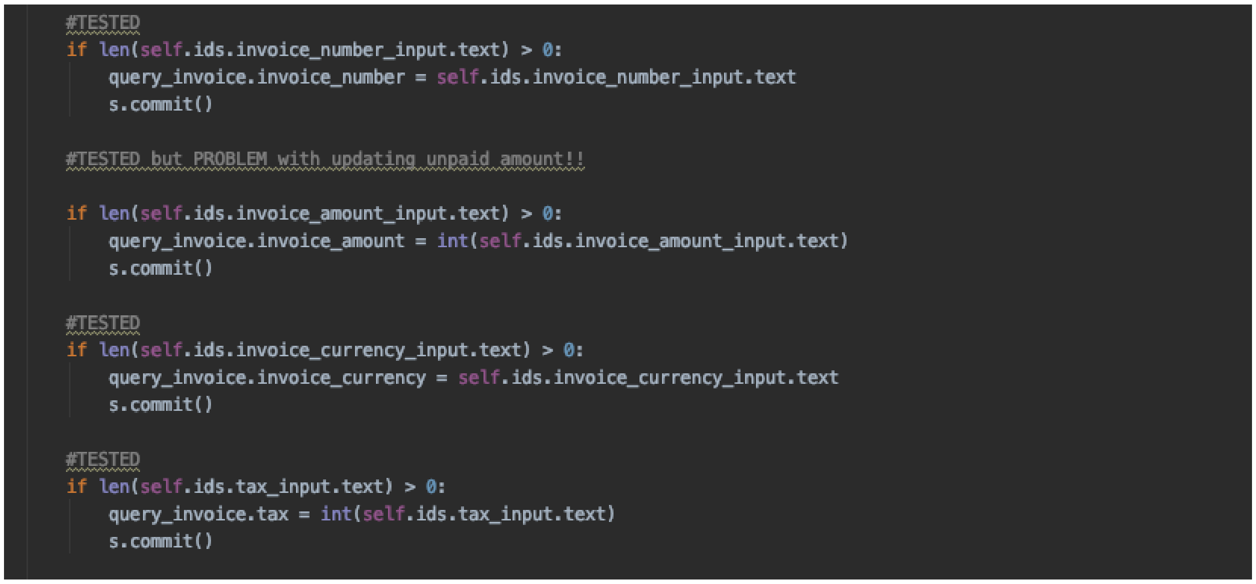
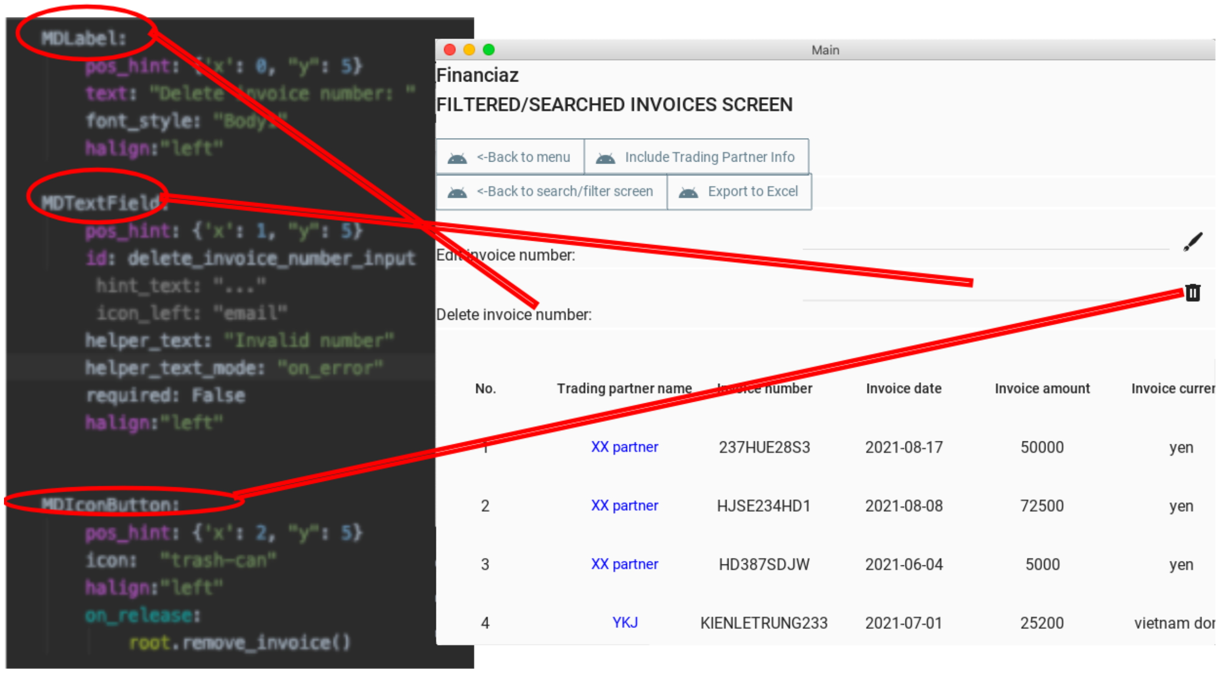

```
    #TESTED
    if len(self.ids.invoice_number_input.text) > 0:
        query_invoice.invoice_number = self.ids.invoice_number_input.text
        s.commit()

    #TESTED but PROBLEM with updating unpaid amount!!

    if len(self.ids.invoice_amount_input.text) > 0:
        query_invoice.invoice_amount = int(self.ids.invoice_amount_input.text)
        s.commit()

    #TESTED
    if len(self.ids.invoice_currency_input.text) > 0:
        query_invoice.invoice_currency = self.ids.invoice_currency_input.text
        s.commit()

    #TESTED
    if len(self.ids.tax_input.text) > 0:
        query_invoice.tax = int(self.ids.tax_input.text)
        s.commit()
```

Figure 25: Storing and updating user input to the added invoice (main.py)

Third, to remove an invoice, the user will type the id of the invoice through a MDTextField on the Filtered searched invoices Screen and click an icon (Figure 26). This will call the function “def remove_invoice()” in the python file and remove the invoice with the input id from the database. (Figure 27)

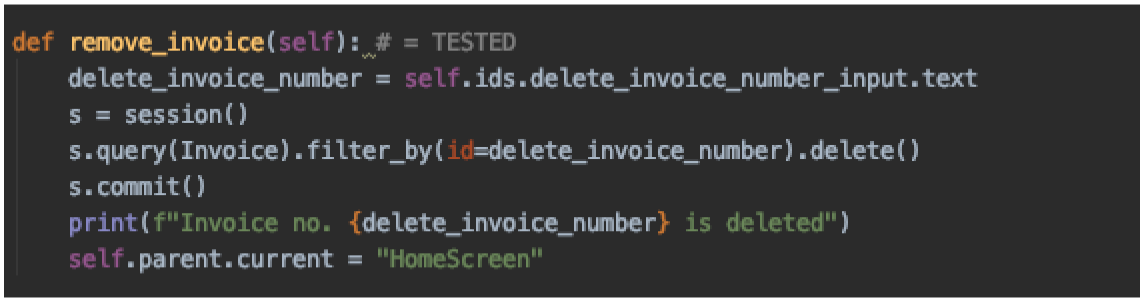


```
MDLabel:
    pos_hint: {'x': 0, "y": 5}
    text: "Delete invoice number: "
    font_style: "Body1"
    halign:"left"

MDTextField:
    pos_hint: {'x': 1, "y": 5}
    id: delete_invoice_number_input
     hint_text: "..."
     icon_left: "email"
    helper_text: "Invalid number"
    helper_text_mode: "on_error"
    required: False
    halign:"left"


MDIconButton:
    pos_hint: {'x': 2, "y": 5}
    icon:  "trash-can"
    halign:"left"
    on_release:
        root.remove_invoice()
```

Main
Financiaz
FILTERED/SEARCHED INVOICES SCREEN
<-Back to menu
Include Trading Partner Info
<-Back to search/filter screen
Export to Excel
Edit invoice number:
Delete invoice number:

No.	Trading partner name	Invoice number	Invoice date	Invoice amount	Invoice curren
1	XX partner	237HUE28S3	2021-08-17	50000	yen
2	XX partner	HJSE234HD1	2021-08-08	72500	yen
3	XX partner	HD387SDJW	2021-06-04	5000	yen
4	YKJ	KIENLETRUNG233	2021-07-01	25200	vietnam dor

Figure 26: Showing the Filtered searched invoices Screen’s GUI and its backend Kivy components in main.kv

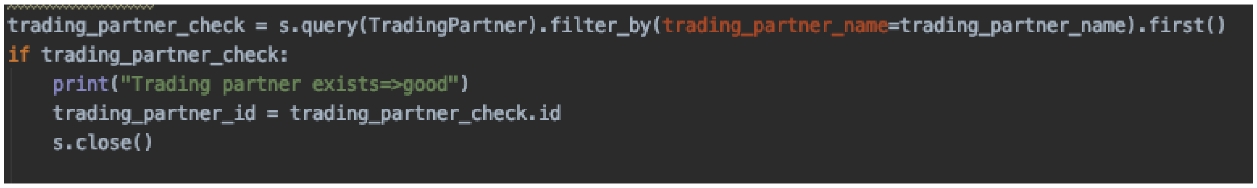


```
def remove_invoice(self): # = TESTED
    delete_invoice_number = self.ids.delete_invoice_number_input.text
    s = session()
    s.query(Invoice).filter_by(id=delete_invoice_number).delete()
    s.commit()
    print(f"Invoice no. {delete_invoice_number} is deleted")
    self.parent.current = "HomeScreen"
```

Figure 27: Showing the def remove_invoice() function for the Filtered searched invoices Screen in the back end (main.py)

4.Program lets the client to input fixed data columns about the Trading Partner and pre-stored data in these fixed data columns for every subsequent invoices added to the program after the first time & Being able to view all data columns of each invoices save in the column

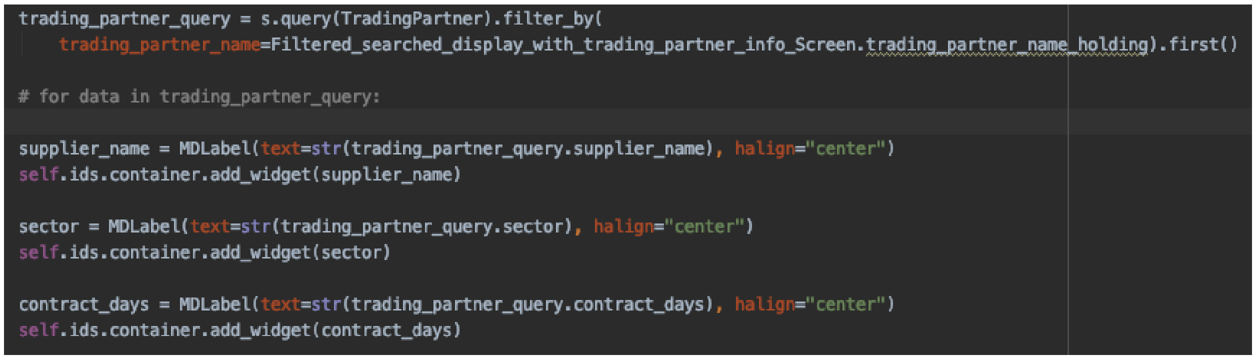
In order to meet this criteria, I coded the program so that it only lets the user add an invoice if the trading partner name they add for the invoice is already registered:



```python
trading_partner_check = s.query(TradingPartner).filter_by(trading_partner_name=trading_partner_name).first()
if trading_partner_check:
    print("Trading partner exists=>good")
    trading_partner_id = trading_partner_check.id
    s.close()
```

Figure 28: Trading partner checking algorithm (main.py)

With this way, when I need to show all the columns/info of an invoice (the info of the trading partner and individual info of an invoice), I can simply query the info of the trading partner that has the same trading partner name with the trading_partner_name value of the invoice and show them along with the info from the invoice (Figure 29). This removes the need for the user to input trading partner info for each invoice every time.

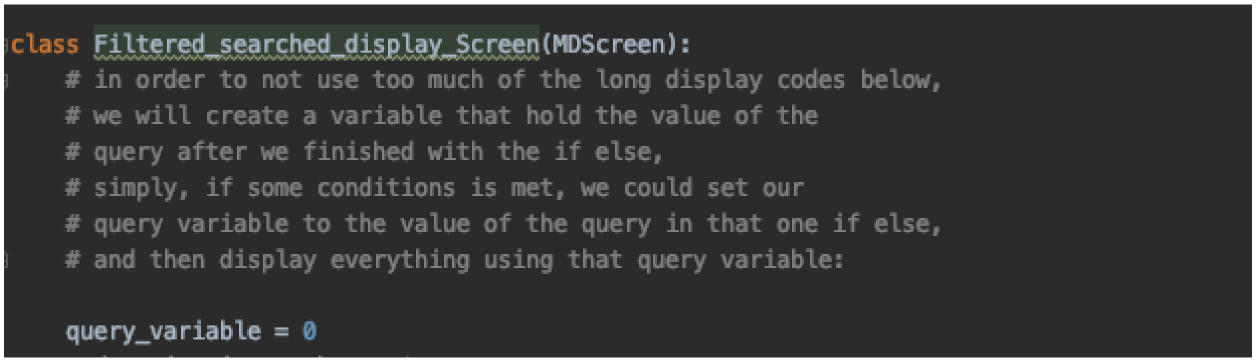


```python
trading_partner_query = s.query(TradingPartner).filter_by(
    trading_partner_name=Filtered_searched_display_with_trading_partner_info_Screen.trading_partner_name_holding).first()

# for data in trading_partner_query:

supplier_name = MDLabel(text=str(trading_partner_query.supplier_name), halign="center")
self.ids.container.add_widget(supplier_name)

sector = MDLabel(text=str(trading_partner_query.sector), halign="center")
self.ids.container.add_widget(sector)

contract_days = MDLabel(text=str(trading_partner_query.contract_days), halign="center")
self.ids.container.add_widget(contract_days)
```

Figure 29: Querying trading partner information of the invoices using its trading partner name (main.py)

5.Let the client filtered/search invoices based on required data columns such as Invoice Number, Trading Partner Name, Invoices added date, Invoices date, Payment information into tables that are viewable on the screen

Depending on the search/filter input from the user, the query for the invoices that will be shown on the Filtered searched display Screen will have to be different. We create a variable to hold different query based on different if statements (reflect the conditions set by the user):

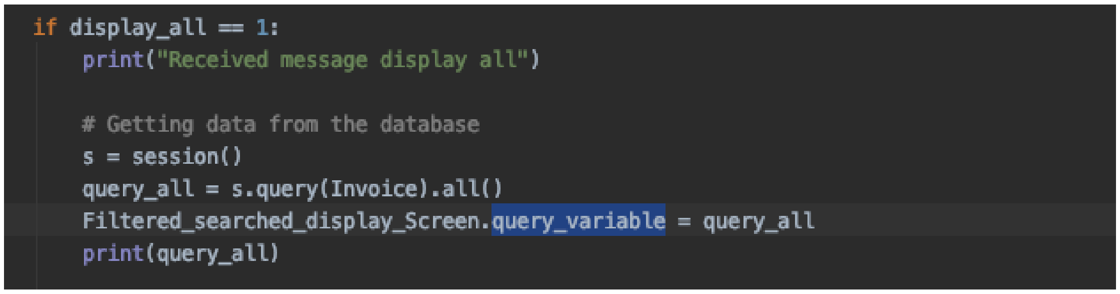


```python
class Filtered_searched_display_Screen(MDScreen):
    # in order to not use too much of the long display codes below,
    # we will create a variable that hold the value of the
    # query after we finished with the if else,
    # simply, if some conditions is met, we could set our
    # query variable to the value of the query in that one if else,
    # and then display everything using that query variable:

    query_variable = 0
```

Figure 30: Setting up the variable to hold the query depends on different conditions (main.py)

Depending on different conditions (different way the user input into the search/filter text fields) , the variable will be changed:

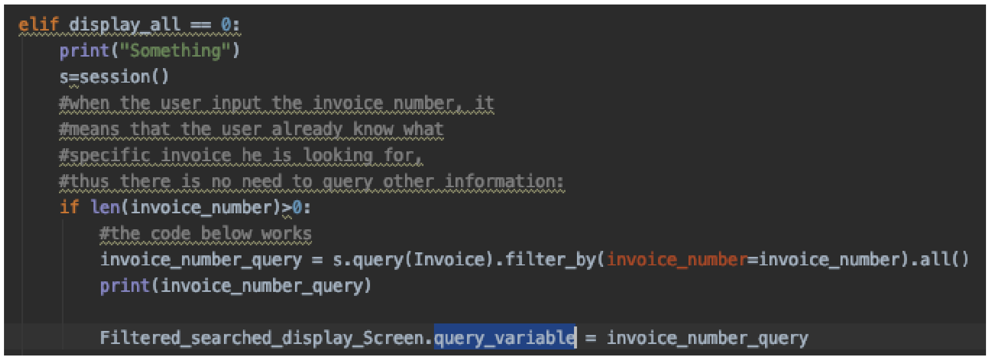
1)



```python
if display_all == 1:
    print("Received message display all")

    # Getting data from the database
    s = session()
    query_all = s.query(Invoice).all()
    Filtered_searched_display_Screen.query_variable = query_all
    print(query_all)
```

Figure 31: When the user wants to display all the invoices (when they didn't input any thing)

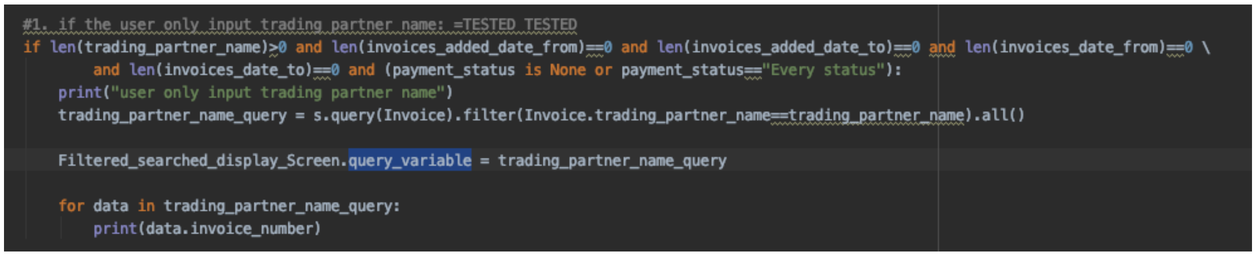
2)



```python
elif display_all == 0:
    print("Something")
    s=session()
    #when the user input the invoice number, it
    #means that the user already know what
    #specific invoice he is looking for,
    #thus there is no need to query other information:
    if len(invoice_number)>0:
        #the code below works
        invoice_number_query = s.query(Invoice).filter_by(invoice_number=invoice_number).all()
        print(invoice_number_query)

        Filtered_searched_display_Screen.query_variable = invoice_number_query
```

Figure 32: When the user input a specific invoice number

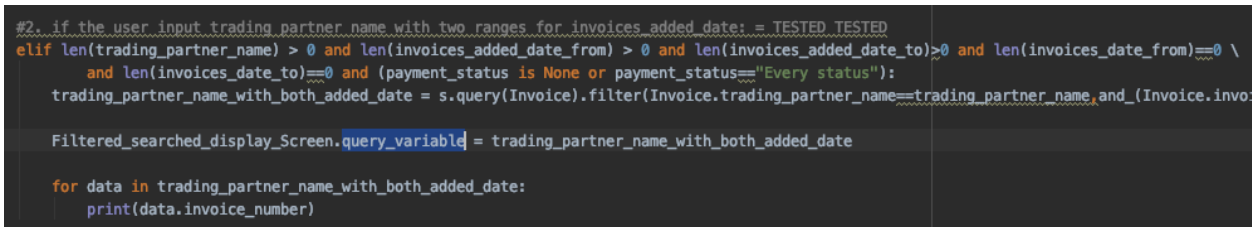
3)



```python
#1. if the user only input trading partner name: =TESTED TESTED
if len(trading_partner_name)>0 and len(invoices_added_date_from)==0 and len(invoices_added_date_to)==0 and len(invoices_date_from)==0 \
        and len(invoices_date_to)==0 and (payment_status is None or payment_status=="Every status"):
    print("user only input trading partner name")
    trading_partner_name_query = s.query(Invoice).filter(Invoice.trading_partner_name==trading_partner_name).all()

    Filtered_searched_display_Screen.query_variable = trading_partner_name_query

    for data in trading_partner_name_query:
        print(data.invoice_number)
```

Figure 33: When the user only input the trading partner name

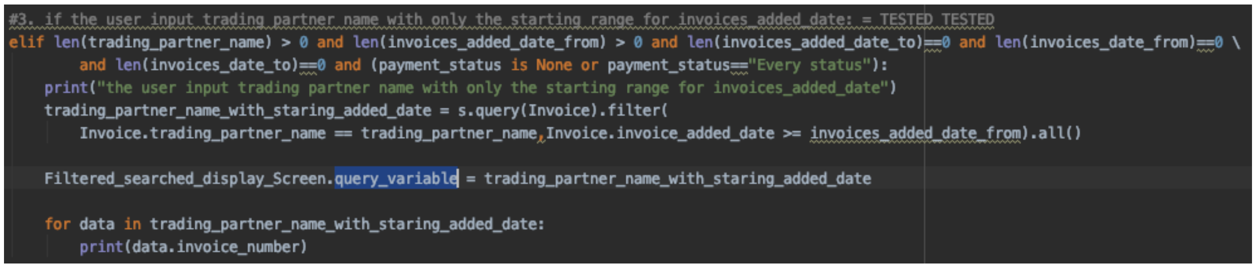
4)



```python
#2. if the user input trading partner name with two ranges for invoices added date: = TESTED TESTED
elif len(trading_partner_name) > 0 and len(invoices_added_date_from) > 0 and len(invoices_added_date_to)>0 and len(invoices_date_from)==0 \
        and len(invoices_date_to)==0 and (payment_status is None or payment_status=="Every status"):
    trading_partner_name_with_both_added_date = s.query(Invoice).filter(Invoice.trading_partner_name==trading_partner_name,and_(Invoice.invo

    Filtered_searched_display_Screen.query_variable = trading_partner_name_with_both_added_date

    for data in trading_partner_name_with_both_added_date:
        print(data.invoice_number)
```

Figure 34: When the user only input the trading partner name and the two ranges for invoices added date

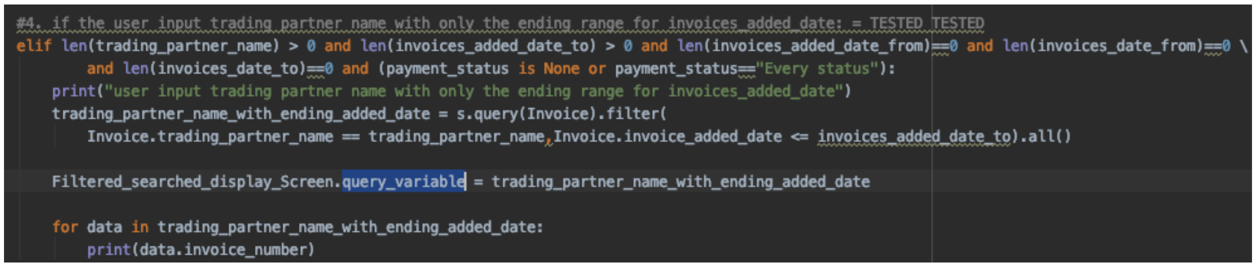
5)



```
#3. if the user input trading partner name with only the starting range for invoices added date: = TESTED TESTED
elif len(trading_partner_name) > 0 and len(invoices_added_date_from) > 0 and len(invoices_added_date_to)==0 and len(invoices_date_from)==0 \
        and len(invoices_date_to)==0 and (payment_status is None or payment_status=="Every status"):
    print("the user input trading partner name with only the starting range for invoices_added_date")
    trading_partner_name_with_staring_added_date = s.query(Invoice).filter(
        Invoice.trading_partner_name == trading_partner_name,Invoice.invoice_added_date >= invoices_added_date_from).all()

    Filtered_searched_display_Screen.query_variable = trading_partner_name_with_staring_added_date

    for data in trading_partner_name_with_staring_added_date:
        print(data.invoice_number)
```

Figure 35: When the user only input the trading partner name with the starting range for invoices added date

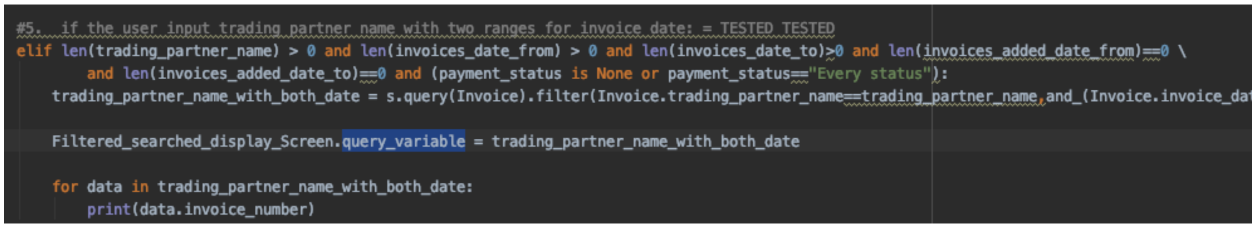
6)



```
#4. if the user input trading partner name with only the ending range for invoices added date: = TESTED TESTED
elif len(trading_partner_name) > 0 and len(invoices_added_date_to) > 0 and len(invoices_added_date_from)==0 and len(invoices_date_from)==0 \
        and len(invoices_date_to)==0 and (payment_status is None or payment_status=="Every status"):
    print("user input trading partner name with only the ending range for invoices_added_date")
    trading_partner_name_with_ending_added_date = s.query(Invoice).filter(
        Invoice.trading_partner_name == trading_partner_name,Invoice.invoice_added_date <= invoices_added_date_to).all()

    Filtered_searched_display_Screen.query_variable = trading_partner_name_with_ending_added_date

    for data in trading_partner_name_with_ending_added_date:
        print(data.invoice_number)
```

Figure 36: When the user only input the trading partner name with the ending range for invoices added date

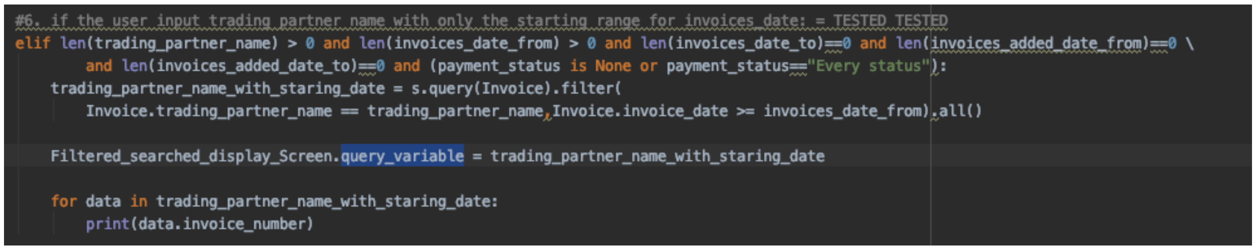
7)



```
#5.  if the user input trading partner name with two ranges for invoice date: = TESTED TESTED
elif len(trading_partner_name) > 0 and len(invoices_date_from) > 0 and len(invoices_date_to)>0 and len(invoices_added_date_from)==0 \
        and len(invoices_added_date_to)==0 and (payment_status is None or payment_status=="Every status"):
    trading_partner_name_with_both_date = s.query(Invoice).filter(Invoice.trading_partner_name==trading_partner_name,and_(Invoice.invoice_dat

    Filtered_searched_display_Screen.query_variable = trading_partner_name_with_both_date

    for data in trading_partner_name_with_both_date:
        print(data.invoice_number)
```

Figure 37: When the user only input the trading partner name and the two ranges for invoice date

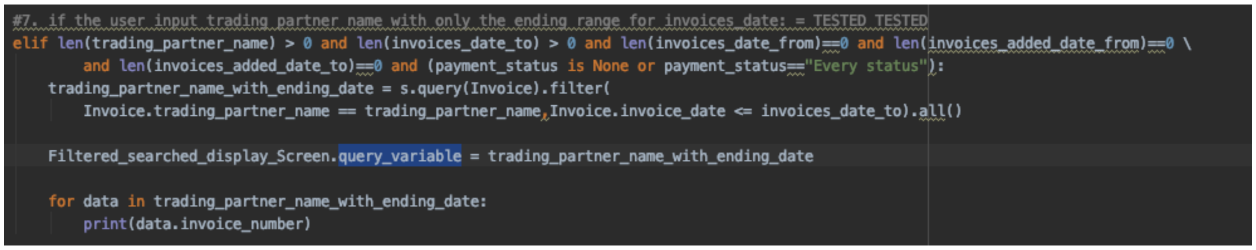
8)



```python
#6. if the user input trading partner name with only the starting range for invoices date: = TESTED TESTED
elif len(trading_partner_name) > 0 and len(invoices_date_from) > 0 and len(invoices_date_to)==0 and len(invoices_added_date_from)==0 \
        and len(invoices_added_date_to)==0 and (payment_status is None or payment_status=="Every status"):
    trading_partner_name_with_staring_date = s.query(Invoice).filter(
        Invoice.trading_partner_name == trading_partner_name,Invoice.invoice_date >= invoices_date_from).all()

    Filtered_searched_display_Screen.query_variable = trading_partner_name_with_staring_date

    for data in trading_partner_name_with_staring_date:
        print(data.invoice_number)
```

Figure 38: When the use only input the trading partner name with the starting range for invoice date

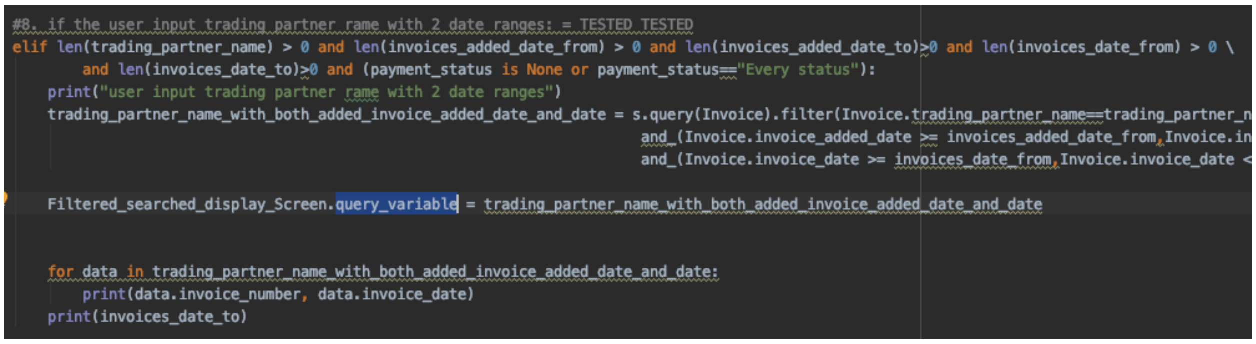
9)



```python
#7. if the user input trading partner name with only the ending range for invoices date: = TESTED TESTED
elif len(trading_partner_name) > 0 and len(invoices_date_to) > 0 and len(invoices_date_from)==0 and len(invoices_added_date_from)==0 \
        and len(invoices_added_date_to)==0 and (payment_status is None or payment_status=="Every status"):
    trading_partner_name_with_ending_date = s.query(Invoice).filter(
        Invoice.trading_partner_name == trading_partner_name,Invoice.invoice_date <= invoices_date_to).all()

    Filtered_searched_display_Screen.query_variable = trading_partner_name_with_ending_date

    for data in trading_partner_name_with_ending_date:
        print(data.invoice_number)
```

Figure 39: When the user only input the trading partner name with the ending range for invoice date

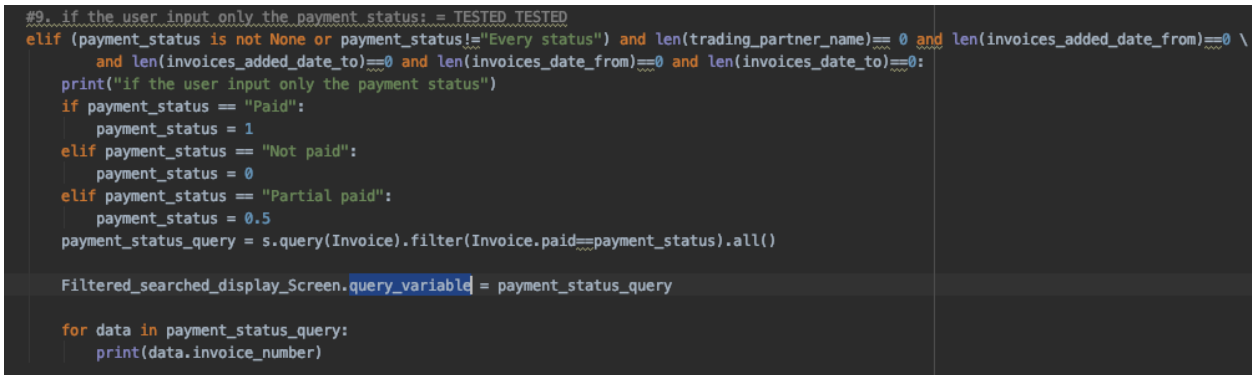
10)



```python
#8. if the user input trading partner rame with 2 date ranges: = TESTED TESTED
elif len(trading_partner_name) > 0 and len(invoices_added_date_from) > 0 and len(invoices_added_date_to)>0 and len(invoices_date_from) > 0 \
        and len(invoices_date_to)>0 and (payment_status is None or payment_status=="Every status"):
    print("user input trading partner rame with 2 date ranges")
    trading_partner_name_with_both_added_invoice_added_date_and_date = s.query(Invoice).filter(Invoice.trading_partner_name==trading_partner_n
                                                                                         and_(Invoice.invoice_added_date >= invoices_added_date_from,Invoice.in
                                                                                         and_(Invoice.invoice_date >= invoices_date_from,Invoice.invoice_date <

    Filtered_searched_display_Screen.query_variable = trading_partner_name_with_both_added_invoice_added_date_and_date


    for data in trading_partner_name_with_both_added_invoice_added_date_and_date:
        print(data.invoice_number, data.invoice_date)
    print(invoices_date_to)
```

Figure 40: When the user only input the 2 ranges for invoice date

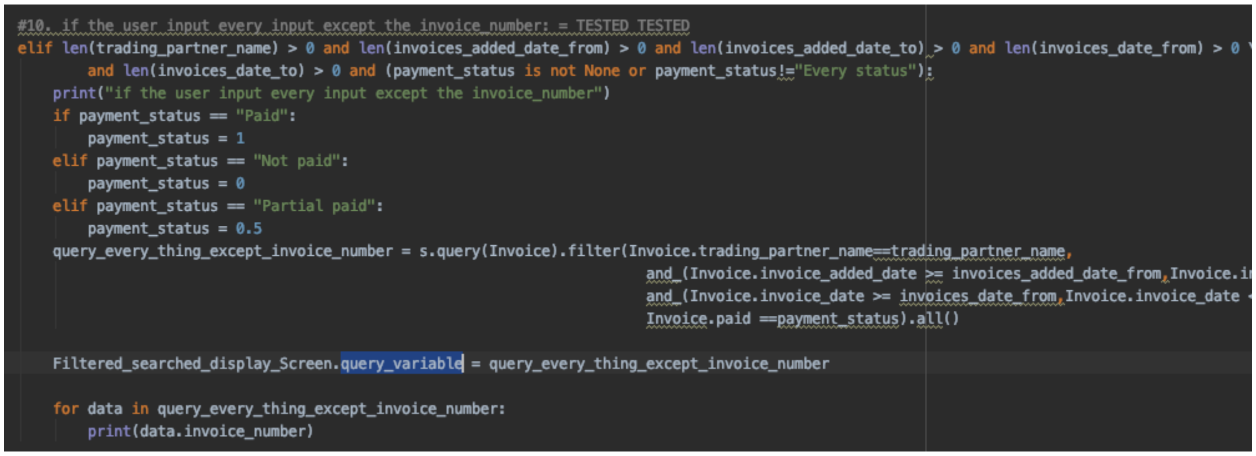
11)



```python
#9. if the user input only the payment status: = TESTED TESTED
elif (payment_status is not None or payment_status!="Every status") and len(trading_partner_name)== 0 and len(invoices_added_date_from)==0 \
        and len(invoices_added_date_to)==0 and len(invoices_date_from)==0 and len(invoices_date_to)==0:
    print("if the user input only the payment status")
    if payment_status == "Paid":
        payment_status = 1
    elif payment_status == "Not paid":
        payment_status = 0
    elif payment_status == "Partial paid":
        payment_status = 0.5
    payment_status_query = s.query(Invoice).filter(Invoice.paid==payment_status).all()

    Filtered_searched_display_Screen.query_variable = payment_status_query

    for data in payment_status_query:
        print(data.invoice_number)
```

Figure 41: When the user only input the payment status

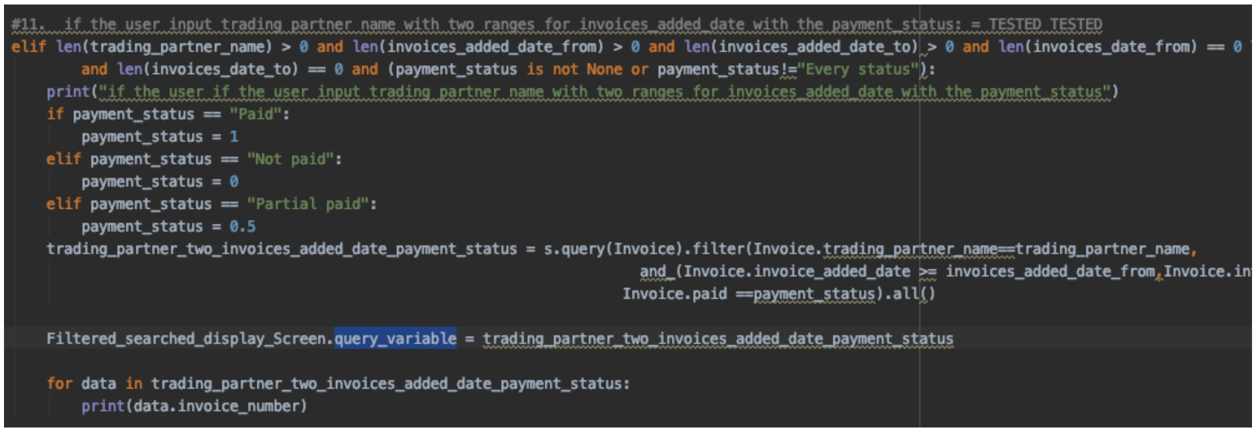
12)



```python
#10. if the user input every input except the invoice_number: = TESTED TESTED
elif len(trading_partner_name) > 0 and len(invoices_added_date_from) > 0 and len(invoices_added_date_to) > 0 and len(invoices_date_from) > 0
        and len(invoices_date_to) > 0 and (payment_status is not None or payment_status!="Every status"):
    print("if the user input every input except the invoice_number")
    if payment_status == "Paid":
        payment_status = 1
    elif payment_status == "Not paid":
        payment_status = 0
    elif payment_status == "Partial paid":
        payment_status = 0.5
    query_every_thing_except_invoice_number = s.query(Invoice).filter(Invoice.trading_partner_name==trading_partner_name,
                                                                       and_(Invoice.invoice_added_date >= invoices_added_date_from,Invoice.i
                                                                       and_(Invoice.invoice_date >= invoices_date_from,Invoice.invoice_date
                                                                       Invoice.paid ==payment_status).all()

    Filtered_searched_display_Screen.query_variable = query_every_thing_except_invoice_number

    for data in query_every_thing_except_invoice_number:
        print(data.invoice_number)
```

Figure 42: When the user input every input except the invoice number

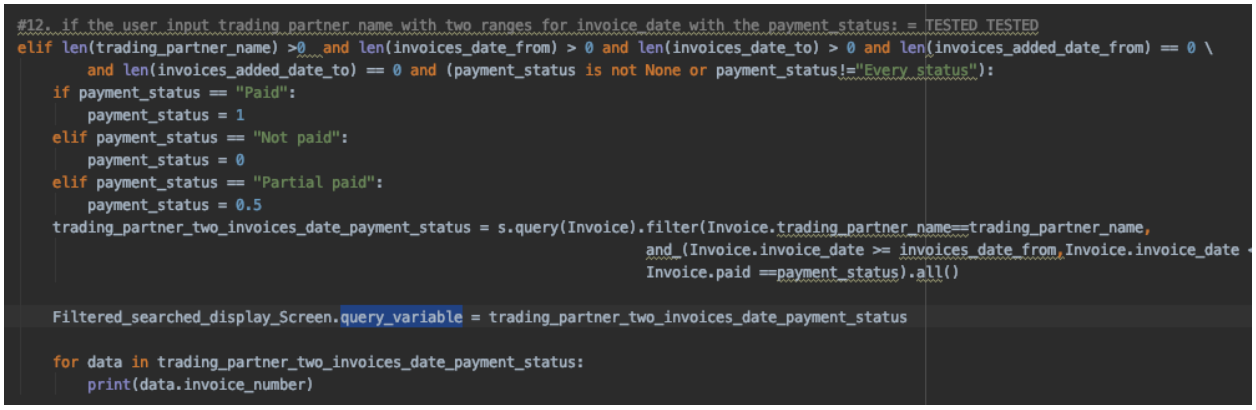
13)



```
#11.  if the user input trading partner name with two ranges for invoices added date with the payment status: = TESTED TESTED
elif len(trading_partner_name) > 0 and len(invoices_added_date_from) > 0 and len(invoices_added_date_to) > 0 and len(invoices_date_from) == 0
        and len(invoices_date_to) == 0 and (payment_status is not None or payment_status!="Every status"):
    print("if the user if the user input trading partner name with two ranges for invoices added date with the payment status")
    if payment_status == "Paid":
        payment_status = 1
    elif payment_status == "Not paid":
        payment_status = 0
    elif payment_status == "Partial paid":
        payment_status = 0.5
    trading_partner_two_invoices_added_date_payment_status = s.query(Invoice).filter(Invoice.trading_partner_name==trading_partner_name,
                                                                   and_(Invoice.invoice_added_date >= invoices_added_date_from,Invoice.in
                                                                  Invoice.paid ==payment_status).all()

    Filtered_searched_display_Screen.query_variable = trading_partner_two_invoices_added_date_payment_status

    for data in trading_partner_two_invoices_added_date_payment_status:
        print(data.invoice_number)
```

Figure 43: When the user only input the trading partner name, two ranges for invoices added date, and the payment status

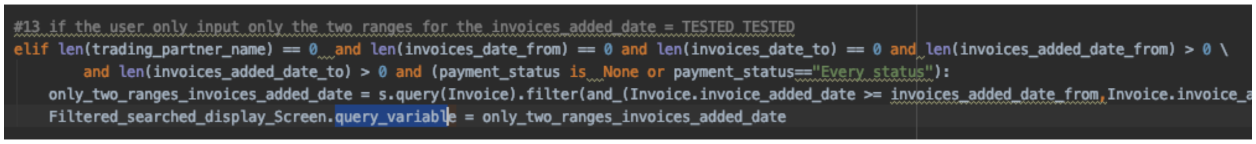
14)



```
#12. if the user input trading partner name with two ranges for invoice date with the payment status: = TESTED TESTED
elif len(trading_partner_name) >0  and len(invoices_date_from) > 0 and len(invoices_date_to) > 0 and len(invoices_added_date_from) == 0 \
        and len(invoices_added_date_to) == 0 and (payment_status is not None or payment_status!="Every status"):
    if payment_status == "Paid":
        payment_status = 1
    elif payment_status == "Not paid":
        payment_status = 0
    elif payment_status == "Partial paid":
        payment_status = 0.5
    trading_partner_two_invoices_date_payment_status = s.query(Invoice).filter(Invoice.trading_partner_name==trading_partner_name,
                                                                   and_(Invoice.invoice_date >= invoices_date_from,Invoice.invoice_date
                                                                   Invoice.paid ==payment_status).all()

    Filtered_searched_display_Screen.query_variable = trading_partner_two_invoices_date_payment_status

    for data in trading_partner_two_invoices_date_payment_status:
        print(data.invoice_number)
```

Figure 44: When the user only input the trading partner name, two ranges for invoice date, and the payment status

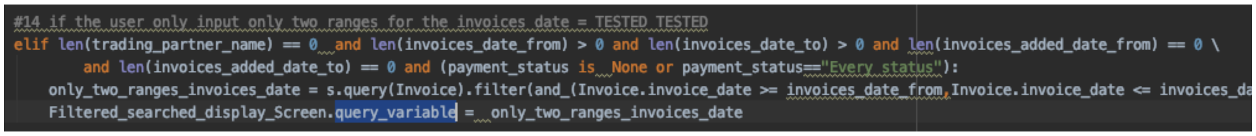
15)



```
#13 if the user only input only the two ranges for the invoices added date = TESTED TESTED
elif len(trading_partner_name) == 0  and len(invoices_date_from) == 0 and len(invoices_date_to) == 0 and len(invoices_added_date_from) > 0 \
        and len(invoices_added_date_to) > 0 and (payment_status is  None or payment_status=="Every status"):
    only_two_ranges_invoices_added_date = s.query(Invoice).filter(and_(Invoice.invoice_added_date >= invoices_added_date_from,Invoice.invoice_a
    Filtered_searched_display_Screen.query_variable = only_two_ranges_invoices_added_date
```

Figure 45: When the user only input the 2 ranges for the invoices added date

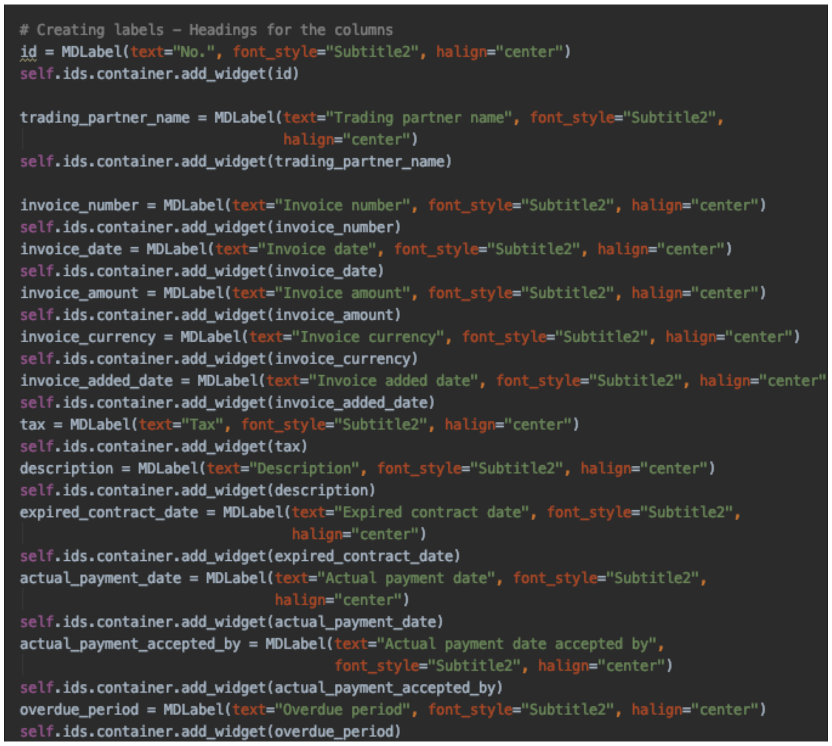
16)



```
#14 if the user only input only two ranges for the invoices date = TESTED TESTED
elif len(trading_partner_name) == 0  and len(invoices_date_from) > 0 and len(invoices_date_to) > 0 and len(invoices_added_date_from) == 0 \
        and len(invoices_added_date_to) == 0 and (payment_status is  None or payment_status=="Every status"):
    only_two_ranges_invoices_date = s.query(Invoice).filter(and_(Invoice.invoice_date >= invoices_date_from,Invoice.invoice_date <= invoices_da
    Filtered_searched_display_Screen.query_variable =  only_two_ranges_invoices_date
```

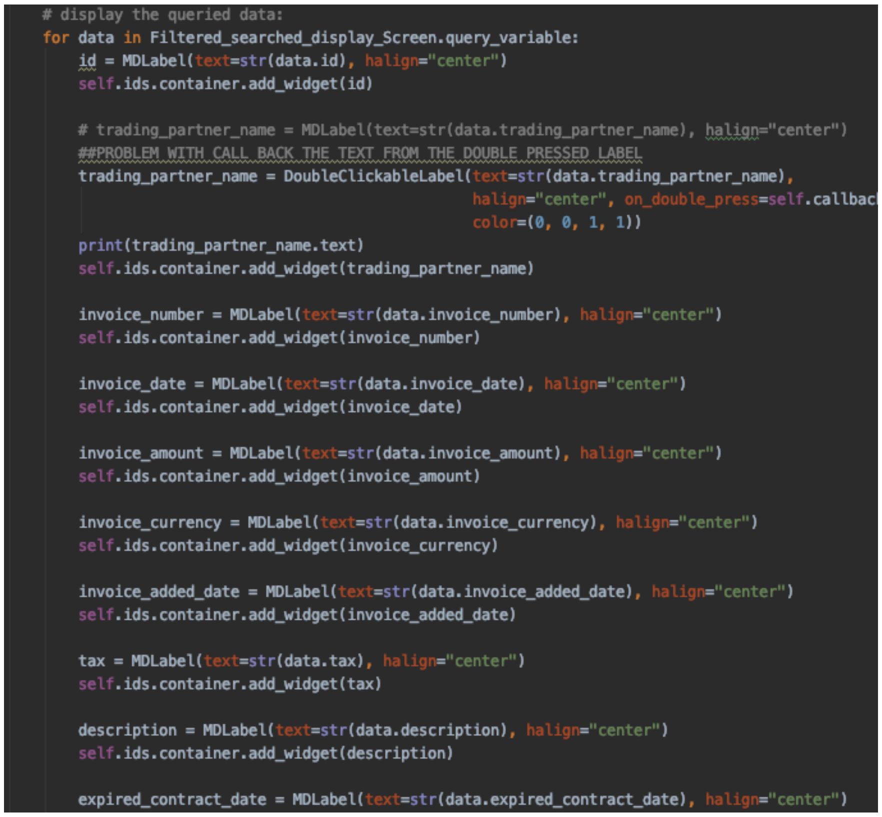
Figure 46: When the user only input the 2 ranges for the invoice date

With this variable, I remove the need to code 16 same lengthy front end codes (Figure 47 & 48) for the 16 different conditions above and instead only have to do it 1 time.



```
# Creating labels - Headings for the columns
id = MDLabel(text="No.", font_style="Subtitle2", halign="center")
self.ids.container.add_widget(id)

trading_partner_name = MDLabel(text="Trading partner name", font_style="Subtitle2",
                               halign="center")
self.ids.container.add_widget(trading_partner_name)

invoice_number = MDLabel(text="Invoice number", font_style="Subtitle2", halign="center")
self.ids.container.add_widget(invoice_number)
invoice_date = MDLabel(text="Invoice date", font_style="Subtitle2", halign="center")
self.ids.container.add_widget(invoice_date)
invoice_amount = MDLabel(text="Invoice amount", font_style="Subtitle2", halign="center")
self.ids.container.add_widget(invoice_amount)
invoice_currency = MDLabel(text="Invoice currency", font_style="Subtitle2", halign="center")
self.ids.container.add_widget(invoice_currency)
invoice_added_date = MDLabel(text="Invoice added date", font_style="Subtitle2", halign="center"
self.ids.container.add_widget(invoice_added_date)
tax = MDLabel(text="Tax", font_style="Subtitle2", halign="center")
self.ids.container.add_widget(tax)
description = MDLabel(text="Description", font_style="Subtitle2", halign="center")
self.ids.container.add_widget(description)
expired_contract_date = MDLabel(text="Expired contract date", font_style="Subtitle2",
                                halign="center")
self.ids.container.add_widget(expired_contract_date)
actual_payment_date = MDLabel(text="Actual payment date", font_style="Subtitle2",
                              halign="center")
self.ids.container.add_widget(actual_payment_date)
actual_payment_accepted_by = MDLabel(text="Actual payment date accepted by",
                                     font_style="Subtitle2", halign="center")
self.ids.container.add_widget(actual_payment_accepted_by)
overdue_period = MDLabel(text="Overdue period", font_style="Subtitle2", halign="center")
self.ids.container.add_widget(overdue_period)
```

Figure 47: Lengthy front end codes no.1 (main.py)

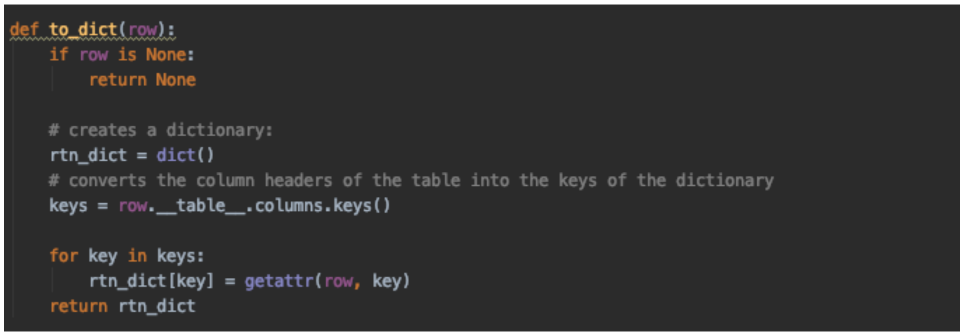


```
# display the queried data:
for data in Filtered_searched_display_Screen.query_variable:
    id = MDLabel(text=str(data.id), halign="center")
    self.ids.container.add_widget(id)

    # trading_partner_name = MDLabel(text=str(data.trading_partner_name), halign="center")
    ##PROBLEM WITH CALL BACK THE TEXT FROM THE DOUBLE PRESSED LABEL
    trading_partner_name = DoubleClickableLabel(text=str(data.trading_partner_name),
                                                halign="center", on_double_press=self.callbac
                                                color=(0, 0, 1, 1))
    print(trading_partner_name.text)
    self.ids.container.add_widget(trading_partner_name)

    invoice_number = MDLabel(text=str(data.invoice_number), halign="center")
    self.ids.container.add_widget(invoice_number)

    invoice_date = MDLabel(text=str(data.invoice_date), halign="center")
    self.ids.container.add_widget(invoice_date)

    invoice_amount = MDLabel(text=str(data.invoice_amount), halign="center")
    self.ids.container.add_widget(invoice_amount)

    invoice_currency = MDLabel(text=str(data.invoice_currency), halign="center")
    self.ids.container.add_widget(invoice_currency)

    invoice_added_date = MDLabel(text=str(data.invoice_added_date), halign="center")
    self.ids.container.add_widget(invoice_added_date)

    tax = MDLabel(text=str(data.tax), halign="center")
    self.ids.container.add_widget(tax)

    description = MDLabel(text=str(data.description), halign="center")
    self.ids.container.add_widget(description)

    expired_contract_date = MDLabel(text=str(data.expired_contract_date), halign="center")
```

Figure 48: Lengthy front end codes no.2 (main.py)

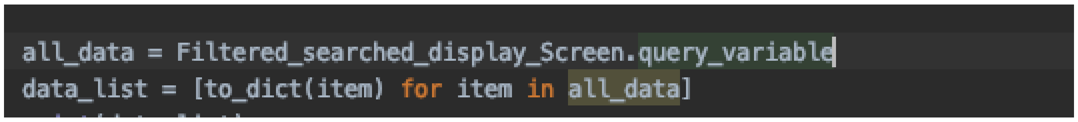
6.Let the client export the filtered/search invoices tables (the invoices don't need to include trading partner data columns) to reports in Excel format. & Let the client export the whole database of invoices to reports in Excel format.

The program turns the query_variable (a variable I talked about in section 6.) into an dictionary with the function “def to_dict()” in order to make it iterable (as it’s easier to add invoices into Excel through a loop) (Figure 49 & 50):



```
def to_dict(row):
    if row is None:
        return None

    # creates a dictionary:
    rtn_dict = dict()
    # converts the column headers of the table into the keys of the dictionary
    keys = row.__table__.columns.keys()

    for key in keys:
        rtn_dict[key] = getattr(row, key)
    return rtn_dict
```

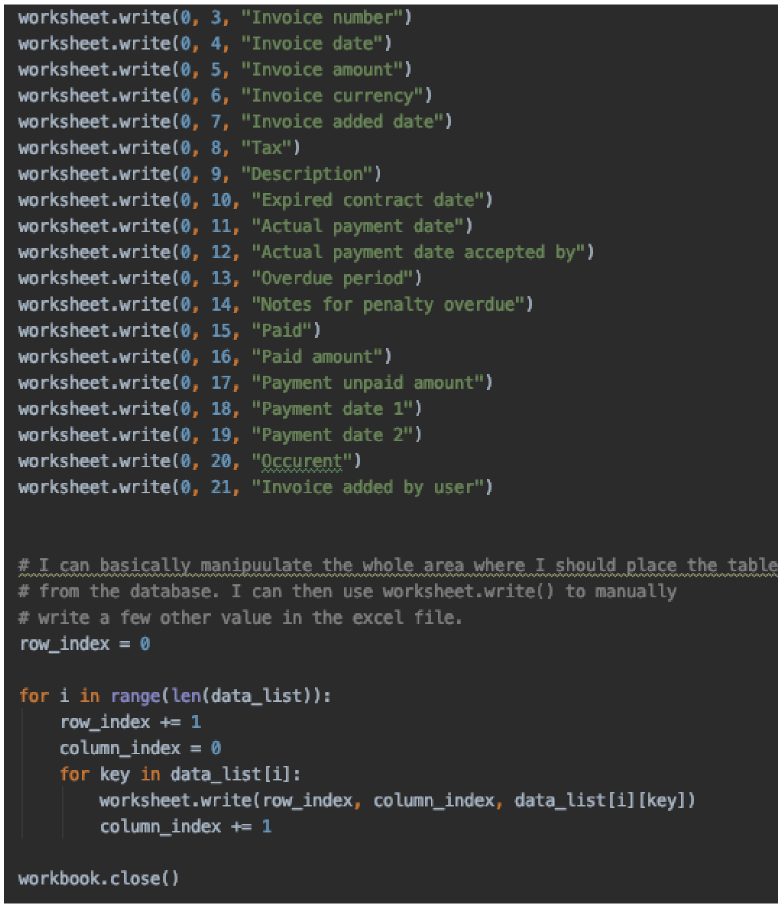
Figure 49: Showing the def to_dict() function (main.py)



```
all_data = Filtered_searched_display_Screen.query_variable
data_list = [to_dict(item) for item in all_data]
```

Figure 50: Showing the algorithm that uses the def to_dict() function to turn a sql query into a dictionary (main.py)

Then, it will create headers for the invoices and add the invoices in the query_variable with a loop to an Excel file:

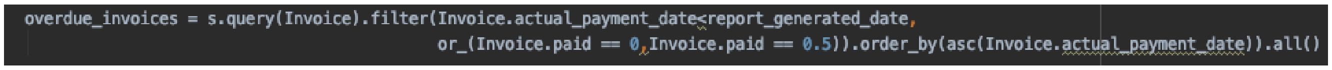


```
worksheet.write(0, 3, "Invoice number")
worksheet.write(0, 4, "Invoice date")
worksheet.write(0, 5, "Invoice amount")
worksheet.write(0, 6, "Invoice currency")
worksheet.write(0, 7, "Invoice added date")
worksheet.write(0, 8, "Tax")
worksheet.write(0, 9, "Description")
worksheet.write(0, 10, "Expired contract date")
worksheet.write(0, 11, "Actual payment date")
worksheet.write(0, 12, "Actual payment date accepted by")
worksheet.write(0, 13, "Overdue period")
worksheet.write(0, 14, "Notes for penalty overdue")
worksheet.write(0, 15, "Paid")
worksheet.write(0, 16, "Paid amount")
worksheet.write(0, 17, "Payment unpaid amount")
worksheet.write(0, 18, "Payment date 1")
worksheet.write(0, 19, "Payment date 2")
worksheet.write(0, 20, "Occurent")
worksheet.write(0, 21, "Invoice added by user")


# I can basically manipuulate the whole area where I should place the table
# from the database. I can then use worksheet.write() to manually
# write a few other value in the excel file.
row_index = 0

for i in range(len(data_list)):
    row_index += 1
    column_index = 0
    for key in data_list[i]:
        worksheet.write(row_index, column_index, data_list[i][key])
        column_index += 1

workbook.close()
```

Figure 51: Adding headers and loop through the dictionary to add the invoices from a query into an Excel file (main.py)

7.Let the client create 2 main types of reports: Overdue Invoice Report and Invoice Payment Schedule Report and export them to reports in Excel format.

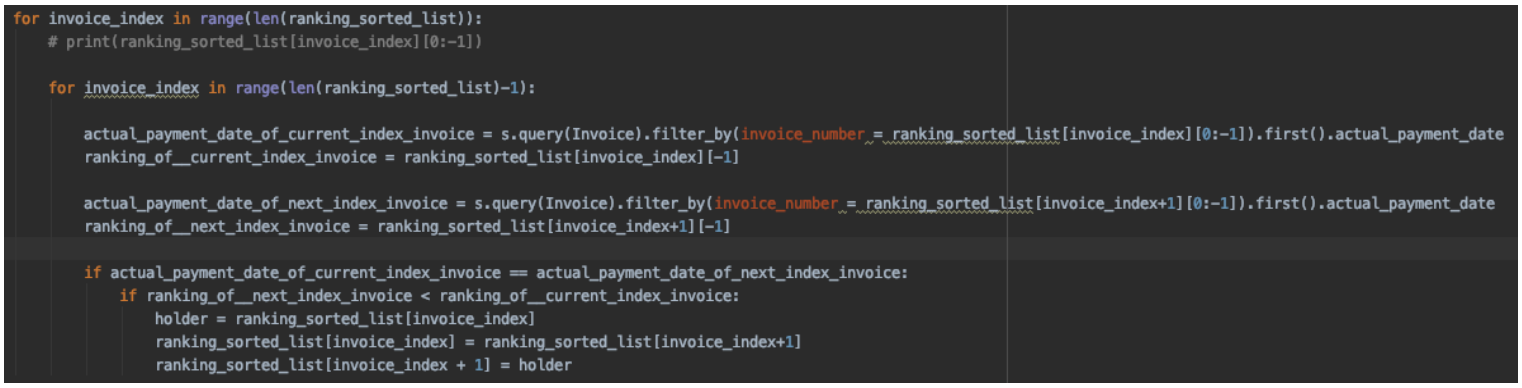
-To create the Overdue Invoice Report, the program do a query that find invoices that has Actual Payment Date before the report generated date and that are not fully paid in the ordered of increasing date:



```
overdue_invoices = s.query(Invoice).filter(Invoice.actual_payment_date<report_generated_date,
                                           or_(Invoice.paid == 0,Invoice.paid == 0.5)).order_by(asc(Invoice.actual_payment_date)).all()
```

Figure 52

And the program will then do a bubble sort to reordered invoices queried based on their ranking (a trading partner value):

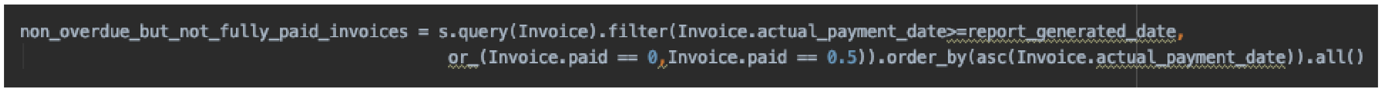


```
for invoice_index in range(len(ranking_sorted_list)):
    # print(ranking_sorted_list[invoice_index][0:-1])

    for invoice_index in range(len(ranking_sorted_list)-1):

        actual_payment_date_of_current_index_invoice = s.query(Invoice).filter_by(invoice_number = ranking_sorted_list[invoice_index][0:-1]).first().actual_payment_date
        ranking_of__current_index_invoice = ranking_sorted_list[invoice_index][-1]

        actual_payment_date_of_next_index_invoice = s.query(Invoice).filter_by(invoice_number = ranking_sorted_list[invoice_index+1][0:-1]).first().actual_payment_date
        ranking_of__next_index_invoice = ranking_sorted_list[invoice_index+1][-1]

        if actual_payment_date_of_current_index_invoice == actual_payment_date_of_next_index_invoice:
            if ranking_of__next_index_invoice < ranking_of__current_index_invoice:
                holder = ranking_sorted_list[invoice_index]
                ranking_sorted_list[invoice_index] = ranking_sorted_list[invoice_index+1]
                ranking_sorted_list[invoice_index + 1] = holder
```

Figure 53: Bubble sort to reordered invoices queried based on their ranking (main.py)

Then the program will do similar code to section **6.** (Figure 49~51) and add the sorted invoices into an Excel file.

-**To create the Invoice Payment Schedule Report,** the program will first adds overdue invoices like above into the Excel file and then query invoices that are not fully paid but has Actual Payment Date equal to or after the report generated date:



```
non_overdue_but_not_fully_paid_invoices = s.query(Invoice).filter(Invoice.actual_payment_date>=report_generated_date,
                                           or_(Invoice.paid == 0,Invoice.paid == 0.5)).order_by(asc(Invoice.actual_payment_date)).all()
```

Figure 54

The program will then do a bubble sort to reordered the invoices based on its ranking and add them below the overdue invoices into the Excel file following similar code in section **6.** (Figure 49~51).