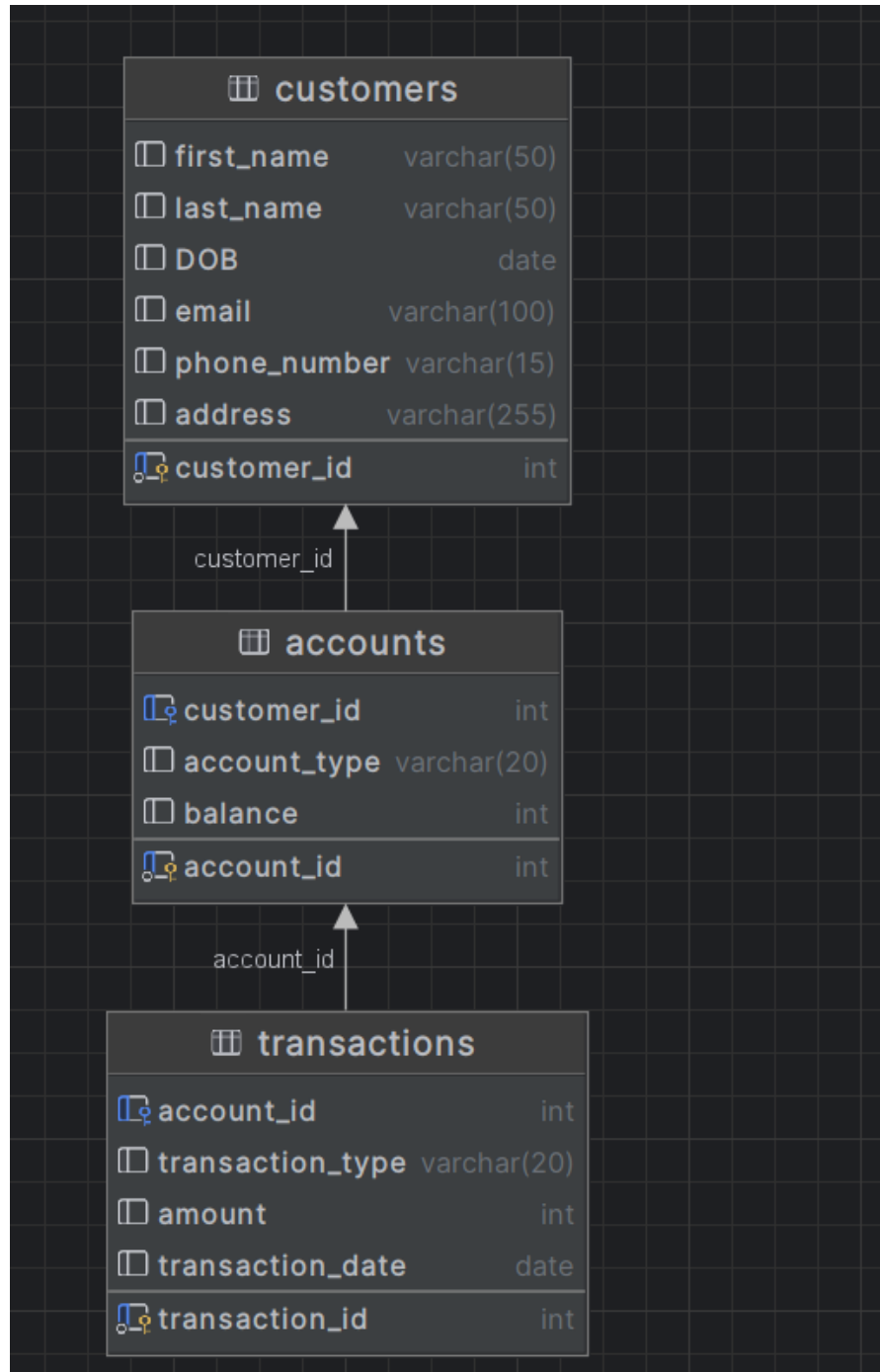
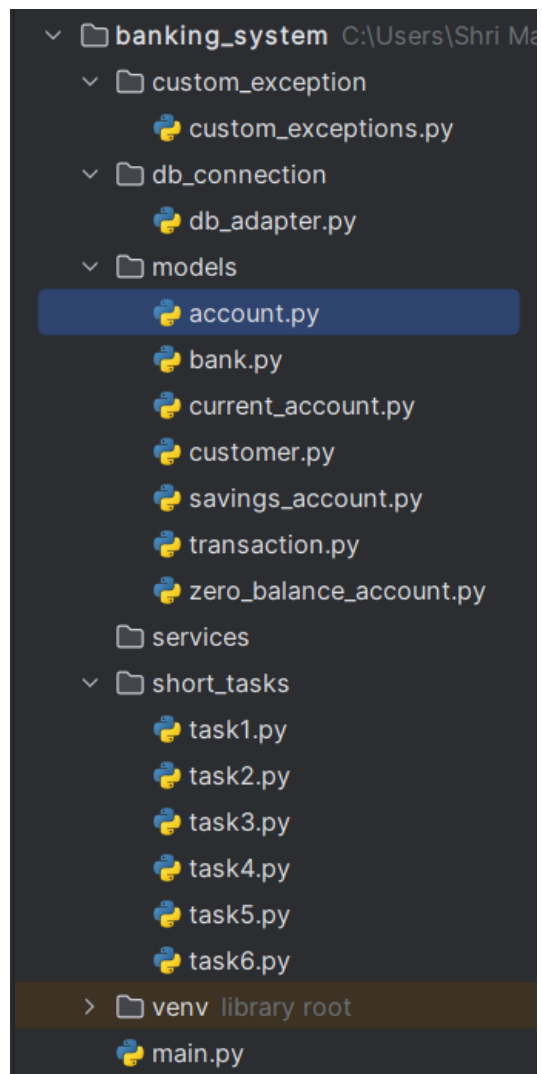


# Banking System

## Database Structure



## File Structure



```
class InsufficientFundsException (Exception):  
    pass  
class InvalidAccountException (Exception):  
    pass  
class OverDraftLimitExceededException(Exception):  
    pass
```

```
config = {  
    'user': 'root',  
    'password': 'Gautam123',  
    'host': 'localhost',  
    'database': 'hmbank'  
}  
  
try:  
    connection = mysql.connector.connect (**config)  
    # print("Connected to the database")  
    return connection  
except mysql.connector.Error as err:  
    print (f"Error: {err}")  
    return None  
  
def get_ids (table_name, id_column_name):  
    mydb = get_db_connection()  
    my_cursor = mydb.cursor()  
    sql = 'SELECT ' + id_column_name + ' FROM ' + table_name + ' ORDER BY ' + id_column_name + ' DESC LIMIT 1'  
    print(sql)  
    my_cursor.execute (sql)  
    x= list (my_cursor.fetchone ()) [0]  
    return int(x)+1
```

Account.py

```

from db_connection.db_adapter import *

class Account:

    def __init__(self, account_id, customer_id, account_type, balance):
        self.connection = get_db_connection()
        self.__account_id = account_id
        self.__customer_id = customer_id
        self.__account_type = account_type
        self.__balance = balance

    def __str__(self):
        return f"Account ID: {self.__account_id}\n" \
               f"Customer ID: {self.__customer_id}\n" \
               f"Account Type: {self.__account_type}\n" \
               f"Balance: ${self.__balance:.2f}"

    def get_account_id(self):
        return self.__account_id

    def get_customer_id(self):
        return self.__customer_id

    def get_account_type(self):
        return self.__account_type

    def get_balance(self):
        return self.__balance

    def update_account_details(self, account_type=None, balance=None):
        my_cursor = self.connection.cursor()

        if account_type:
            sql = '''
                UPDATE Accounts SET account_type = %s WHERE account_id = %s
            '''
            para = (account_type, self.__account_id)
            my_cursor.execute(sql, para)
            self.connection.commit()
            self.__account_type = account_type
            print('Account Type updated successfully')

        if balance:
            sql = '''
                UPDATE Accounts SET balance = %s WHERE account_id = %s
            '''
            para = (balance, self.__account_id)
            my_cursor.execute(sql, para)

```

```

        self.connection.commit()
        self.__balance = balance
        print('Account Type updated successfully')

def deposit(self, amount):
    try:
        my_cursor = self.connection.cursor()
        sql = '''
            UPDATE Accounts SET balance = %s WHERE account_id = %s
            '''
        para = (self.__balance + amount, self.__account_id)
        my_cursor.execute(sql, para)
        self.connection.commit()
        self.__balance += amount
        print('Amount deposited successfully')
    except Exception as e:
        print(f'An error occurred: {e}')

def withdraw(self, amount):
    if amount > self.__balance:
        print('Insufficient balance')
        return
    try:
        my_cursor = self.connection.cursor()
        sql = '''
            UPDATE Accounts SET balance = %s WHERE account_id = %s
            '''
        para = (self.__balance - amount, self.__account_id)
        my_cursor.execute(sql, para)
        self.connection.commit()
        self.__balance -= amount
        print('Amount withdrawn successfully')
    except Exception as e:
        print(f'An error occurred: {e}')

def calculate_interest(self):
    print(f'Amount after interest: ${self.__balance + (self.__balance *
0.45)}')
    return self.__balance + (self.__balance * 0.45)

def print_account_info(self):
    print('Account ID:', self.__account_id)
    print('Account Type:', self.__account_type)
    print('Account Balance:', self.__balance)

```

Bank.py

```

from db_connection.db_adapter import *

```

```

from models.account import Account
from models.savings_account import SavingsAccount
from models.current_account import CurrentAccount
from models.transaction import Transaction

class Bank:

    def __init__(self):
        self.connection = get_db_connection()

    def deposit(self, account_id, amount):
        my_cursor = self.connection.cursor()
        try:
            sql = '''
                UPDATE Accounts SET balance = balance + %s WHERE account_id = %s
            '''
            para = (amount, account_id)
            my_cursor.execute(sql, para)
            self.connection.commit()
            print('Amount deposited successfully')
        except Exception as e:
            print(f'An error occurred: {e}')

    def withdraw(self, account_id, amount):
        my_cursor = self.connection.cursor()
        try:
            sql = '''
                UPDATE Accounts SET balance = balance - %s WHERE account_id = %s
            '''
            para = (amount, account_id)
            my_cursor.execute(sql, para)
            self.connection.commit()
            print('Amount withdrawn successfully')
        except Exception as e:
            print(f'An error occurred: {e}')

    def get_account_by_id(self, account_id):
        try:
            my_cursor = self.connection.cursor()
            sql = '''
                SELECT * FROM Accounts WHERE account_id = %s
            '''
            para = (account_id,)
            my_cursor.execute(sql, para)
            x = Account(*list(my_cursor.fetchone()))
            return x
        except Exception as e:
            print(f'An error occurred: {e}')

```

```

def calculate_interest(self, account_id):
    try:
        customer_account = self.get_account_by_id(account_id)
        value = customer_account.calculate_interest()
        customer_account.update_account_details(balance=value)
    except Exception as e:
        print(f'An error occurred: {e}')

def create_customer_account(self, account):
    try:
        my_cursor = self.connection.cursor()
        sql = '''
            INSERT INTO Accounts(account_id, customer_id, account_type,
balance)
            VALUES (%s, %s, %s, %s)
            '''
        para = (account.get_account_id(), account.get_customer_id(),
account.get_account_type(), account.get_balance())
        my_cursor.execute(sql, para)
        self.connection.commit()
        print('Account created successfully')
    except Exception as e:
        print(f'An error occurred: {e}')

def create_account(self):
    print("Choose the type of account:")
    print("1. Savings Account")
    print("2. Current Account")

    choice = input("Enter your choice (1 or 2): ")

    # account_id = input("Enter the account ID: ")
    customer_id = input("Enter the customer ID: ")
    initial_balance = float(input("Enter the initial balance: "))

    if choice == "1":
        interest_rate = float(input("Enter the interest rate for Savings
Account: "))
        account = SavingsAccount(get_ids('accounts', 'account_id'),
customer_id, initial_balance, interest_rate)
        self.create_customer_account(account)
    elif choice == "2":
        account = CurrentAccount(get_ids('accounts', 'account_id'),
customer_id, initial_balance)
        self.create_customer_account(account)
    else:
        print("Invalid choice. Please choose 1 or 2.")
        return

```

```

        print("Account created successfully!")
        print("Account details:")
        account.print_account_info()

def get_account_balance_by_id(self, account_id):
    try:
        customer_account = self.get_account_by_id(account_id)
        return customer_account.get_balance()
    except Exception as e:
        print(f'An error occurred: {e}')

def get_account_details(self, account_id):
    try:
        customer_account = self.get_account_by_id(account_id)
        customer_account.print_account_info()
    except Exception as e:
        print(f'An error occurred: {e}')

def transfer(self, sender_account_id, receiver_account_id, amount):
    try:
        sender_account = self.get_account_by_id(sender_account_id)
        receiver_account = self.get_account_by_id(receiver_account_id)

        if sender_account.get_balance() < amount:
            print('Insufficient balance in sender account')
        else:
            sender_account.withdraw(amount)
            receiver_account.deposit(amount)
            print('Transaction made successfully')

    except Exception as e:
        print(f'An error occurred: {e}')

def get_transactions(self, account_id, start_date, end_date):
    try:
        my_cursor = self.connection.cursor()
        sql = '''
            SELECT * FROM Transactions WHERE account_id = %s AND
            transaction_date BETWEEN %s AND %s
        '''
        para = (account_id, start_date, end_date)
        my_cursor.execute(sql, para)
        x = [Transaction(*list(i)) for i in list(my_cursor.fetchall())]
        return x
    except Exception as e:
        print(f'An error occurred: {e}')

def list_all_account(self):

```



```

try:
    my_cursor = self.connection.cursor()
    sql = '''
        SELECT * FROM Accounts
    '''
    my_cursor.execute(sql)
    x = [Account(*list(i)) for i in list(my_cursor.fetchall())]
    return x
except Exception as e:
    print(f'An error occurred: {e}')

```

## Current\_account.py

```

from db_connection.db_adapter import *
from models.account import Account

class CurrentAccount(Account):
    OVERDRAFT_LIMIT = 1000

    def __init__(self, account_id, customer_id, balance):
        super().__init__(account_id, customer_id, account_type="Current",
            balance=balance)
        self.__overdraft_limit = self.OVERDRAFT_LIMIT

    def withdraw(self, amount):
        if amount > super().get_balance() + self.__overdraft_limit:
            print('Withdrawal amount exceeds available balance and overdraft
limit.')
            return

        try:
            my_cursor = super().connection.cursor()
            sql = '''
                UPDATE Accounts SET balance = %s WHERE account_id = %s
            '''
            para = (super().get_balance() - amount, super().get_account_id())
            my_cursor.execute(sql, para)
            super().connection.commit()
            super().update_account_details(balance=super().get_balance()-amount)
            print('Amount withdrawn successfully')
        except Exception as e:
            print(f'An error occurred: {e}')

```

## Customer.py

```
from db_connection.db_adapter import *

class Customer:

    def __init__(self, customer_id, first_name, last_name, dob, email,
phone_number, address):
        self.connection = get_db_connection()
        self.__customer_id = customer_id
        self.__first_name = first_name
        self.__last_name = last_name
        self.__dob = dob
        self.__email = email
        self.__phone_number = phone_number
        self.__address = address

    def get_customer_id(self):
        return self.__customer_id

    def get_first_name(self):
        return self.__first_name

    def get_last_name(self):
        return self.__last_name

    def get_customer_email(self):
        return self.__email

    def get_phone_number(self):
        return self.__phone_number

    def get_customer_address(self):
        return self.__address

    def update_student_info(self, first_name=None, last_name=None,
date_of_birth=None, email=None, phone_number=None, address=None):
        my_cursor = self.connection.cursor()

        if first_name:
            sql = '''
                UPDATE Customers SET first_name = %s WHERE customer_id = %s
            '''
            para = (first_name, self.__customer_id)
            my_cursor.execute(sql, para)
            self.connection.commit()
            self.__first_name = first_name
```

```

if last_name:
    sql = '''
        UPDATE Customers SET last_name = %s WHERE customer_id = %s
    '''
    para = (last_name, self.__customer_id)
    my_cursor.execute(sql, para)
    self.connection.commit()
    self.__last_name = last_name

if date_of_birth:
    sql = '''
        UPDATE Customers SET DOB = %s WHERE customer_id = %s
    '''
    para = (date_of_birth, self.__customer_id)
    my_cursor.execute(sql, para)
    self.connection.commit()
    self.__dob = date_of_birth

if email:
    sql = '''
        UPDATE Customers SET email = %s WHERE customer_id = %s
    '''
    para = (email, self.__customer_id)
    my_cursor.execute(sql, para)
    self.connection.commit()
    self.__email = email

if phone_number:
    sql = '''
        UPDATE Customers SET phone_number = %s WHERE customer_id = %s
    '''
    para = (phone_number, self.__customer_id)
    my_cursor.execute(sql, para)
    self.connection.commit()
    self.__phone_number = phone_number

if address:
    sql = '''
        UPDATE Customers SET address = %s WHERE customer_id = %s
    '''
    para = (address, self.__customer_id)
    my_cursor.execute(sql, para)
    self.connection.commit()
    self.__address = address

print('Student Details Updated Successfully')

```

Savings\_account.py

```

from db_connection.db_adapter import *
from models.account import Account

class SavingsAccount(Account):
    def __init__(self, account_id, customer_id, balance, interest_rate):
        super().__init__(account_id, customer_id, account_type="Savings",
balance=balance)
        self.__interest_rate = interest_rate

    def calculate_interest(self):
        interest_amount = super().get_balance() * (self.__interest_rate / 100)
        print(f'Interest calculated for Savings Account:
${interest_amount:.2f}')
        return super().get_balance() + interest_amount

```

## Transaction.py

```

from db_connection.db_adapter import *

class Transaction:

    def __init__(self, transaction_id, account_id, transaction_type, amount,
transaction_date):
        self.connection = get_db_connection()
        self.__transaction_id = transaction_id
        self.__account_id = account_id
        self.__transaction_type = transaction_type
        self.__amount = amount
        self.__transaction_date = transaction_date

    def __str__(self):
        return f"Transaction ID: {self.__transaction_id}\n" \
f"Account ID: {self.__account_id}\n" \
f"Transaction Type: {self.__transaction_type}\n" \
f"Amount: ${self.__amount:.2f}\n" \
f"Transaction Date: {self.__transaction_date}"

    def get_transaction_id(self):
        return self.__transaction_id

    def get_account_id(self):
        return self.__account_id

    def get_transaction_type(self):
        return self.__transaction_type

```

```

def get_transaction_date(self):
    return self.__transaction_date

def get_transaction_amount(self):
    return self.__amount

def update_transaction_info(self, account_id=None, transaction_type=None,
transaction_amount=None,
                           transaction_date=None):

    my_cursor = self.connection.cursor()

    if account_id:
        try:
            sql = '''
UPDATE Transactions SET account_id = %s WHERE
transaction_id = %s
'''
            para = (account_id, self.__transaction_id)
            my_cursor.execute(sql, para)
            self.connection.commit()
            print('Account id updated successfully')
        except Exception as e:
            print(f'An error occurred: {e}')

    if transaction_type:
        try:
            sql = '''
UPDATE Transactions SET transaction_type = %s
WHERE transaction_id = %s
'''
            para = (transaction_type, self.__transaction_id)
            my_cursor.execute(sql, para)
            self.connection.commit()
            print('Transaction type updated successfully')
        except Exception as e:
            print(f'An error occurred: {e}')

    if transaction_amount:
        try:
            sql = '''
UPDATE Transactions SET amount = %s WHERE
transaction_id = %s
'''
            para = (transaction_amount, self.__transaction_id)
            my_cursor.execute(sql, para)
            self.connection.commit()
            print('Transaction Amount updated successfully')
        except Exception as e:

```

```

        print(f'An error occurred: {e}')

    if transaction_date:
        try:
            sql = '''
                UPDATE Transactions SET transaction_date = %s
            WHERE transaction_id = %s
            '''
            para = (transaction_date, self.__transaction_id)
            my_cursor.execute(sql, para)
            self.connection.commit()
            print('Transaction date updated successfully')
        except Exception as e:
            print(f'An error occurred: {e}')

```

## Zero\_balance.py

```

from db_connection.db_adapter import *
from models.account import Account

class ZeroBalanceAccount(Account):

    def __init__(self, account_id, customer_id, account_type):
        self.connection = get_db_connection()
        super().__init__(account_id, customer_id, account_type, 0)

```

## Main.py

```

from models.bank import Bank

class BankApp:
    def __init__(self):
        self.bank = Bank()

    def create_account(self):
        while True:
            print("\nCreate Account Menu:")
            print("1. Enter Account Details")
            print("2. Exit")

            choice = input("Enter your choice (1-3): ")

            if choice == "1":
                self.bank.create_account()

```

```

        elif choice == "2":
            print("Exiting Create Account Menu.")
            break
        else:
            print("Invalid choice. Please choose a valid option (1-3).")

def main(self):
    while True:
        print("\nBank App Menu:")
        print("1. Create Account")
        print("2. Deposit")
        print("3. Withdraw")
        print("4. Get Balance")
        print("5. Transfer")
        print("6. Get Account Details")
        print("7. List Accounts")
        print("8. Get Transactions")
        print("9. Exit")

        choice = input("Enter your choice (1-9): ")

        if choice == "1":
            self.create_account()
        elif choice == "2":
            account_id = input("Enter the account ID: ")
            amount = float(input("Enter the deposit amount: "))
            self.bank.deposit(account_id, amount)
        elif choice == "3":
            account_id = input("Enter the account ID: ")
            amount = float(input("Enter the withdrawal amount: "))
            self.bank.withdraw(account_id, amount)
        elif choice == "4":
            account_id = input("Enter the account ID: ")
            temp_account = self.bank.get_account_by_id(account_id)
            print(temp_account.get_balance())
        elif choice == "5":
            from_account_id = input("Enter the source account ID: ")
            to_account_id = input("Enter the destination account ID: ")
            amount = float(input("Enter the transfer amount: "))
            self.bank.transfer(from_account_id, to_account_id, amount)
        elif choice == "6":
            account_id = input("Enter the account ID: ")
            self.bank.get_account_details(account_id)
        elif choice == "7":
            x = self.bank.list_all_account()
            print(*x, sep="\n\n")
        elif choice == "8":
            account_id = input("Enter the account ID: ")
            start_date = input("Enter Start Date: ")

```

```

        end_date = input("Enter End Date: ")
        x = self.bank.get_transactions(account_id, start_date, end_date)
        print(*x, sep="\n\n")
    elif choice == "9":
        print("Exiting Bank App. Goodbye!")
        break
    else:
        print("Invalid choice. Please choose a valid option (1-9).")

if __name__ == "__main__":
    bank_app = BankApp()
    bank_app.main()

```

Output:

```

Bank App Menu:
1. Create Account
2. Deposit
3. Withdrawal Packages
4. Get Balance
5. Transfer
6. Get Account Details
7. List Accounts
8. Get Transactions
9. Exit
Enter your choice (1-9):

```



```
6. Get Account Details
7. List Accounts
8. Get Transactions
9. Exit
Enter your choice (1-9): 7
Account ID: 101
Customer ID: 1
Account Type: savings
Balance: $2355.00

Account ID: 102
Customer ID: 2
Account Type: current
Balance: $13000.00

Account ID: 103
Customer ID: 3
Account Type: savings
Balance: $7350.00

Account ID: 104
Customer ID: 4
Account Type: current
```

```
Enter your choice (1-9): 6
Enter the account ID: 101
Account ID: 101
Account Type: savings
Account Balance: 2355
```

```
Enter your choice (1-9): 8
Enter the account ID: 101
Enter Start Date: 2023-01-10
Enter End Date: 2023-08-10
```

```
Enter your choice (1-9): 4
Enter the account ID: 101
2355
```

```
Enter your choice (1-9): 2
Enter the account ID: 101
Enter the deposit amount: 100
Amount deposited successfully
```

```
Enter your choice (1-9): 3
Enter the account ID: 101
Enter the withdrawal amount: 300
Amount withdrawn successfully
```

```
Enter your choice (1-9): 5
Enter the source account ID: 101
Enter the destination account ID: 102
Enter the transfer amount: 100
Amount withdrawn successfully
Amount deposited successfully
Transaction made successfully
```

Bank App Menu:

1. Create Account
2. Deposit
3. Withdraw
4. Get Balance
5. Transfer
6. Get Account Details
7. List Accounts
8. Get Transactions
9. Exit

Enter your choice (1-9): 9

Exiting Bank App. Goodbye!