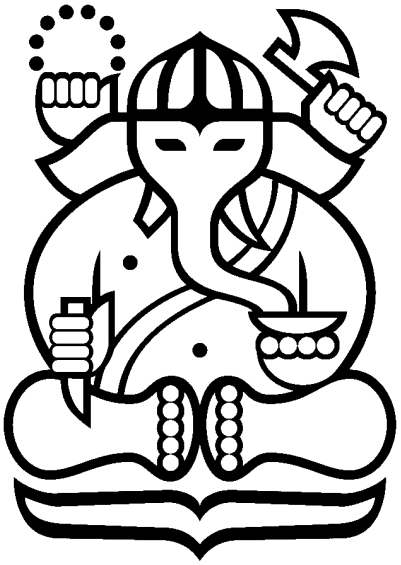
TUGAS BESAR: SWALAYAN
IF5021 ALGORITMA DAN PEMROGRAMAN B



Disusun oleh:

Michael Hans	13518056
Byan Sakura Kireyna Aji	13518066

Tanggal Pengumpulan
Jumat, 3 Desember 2021

Program Studi Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

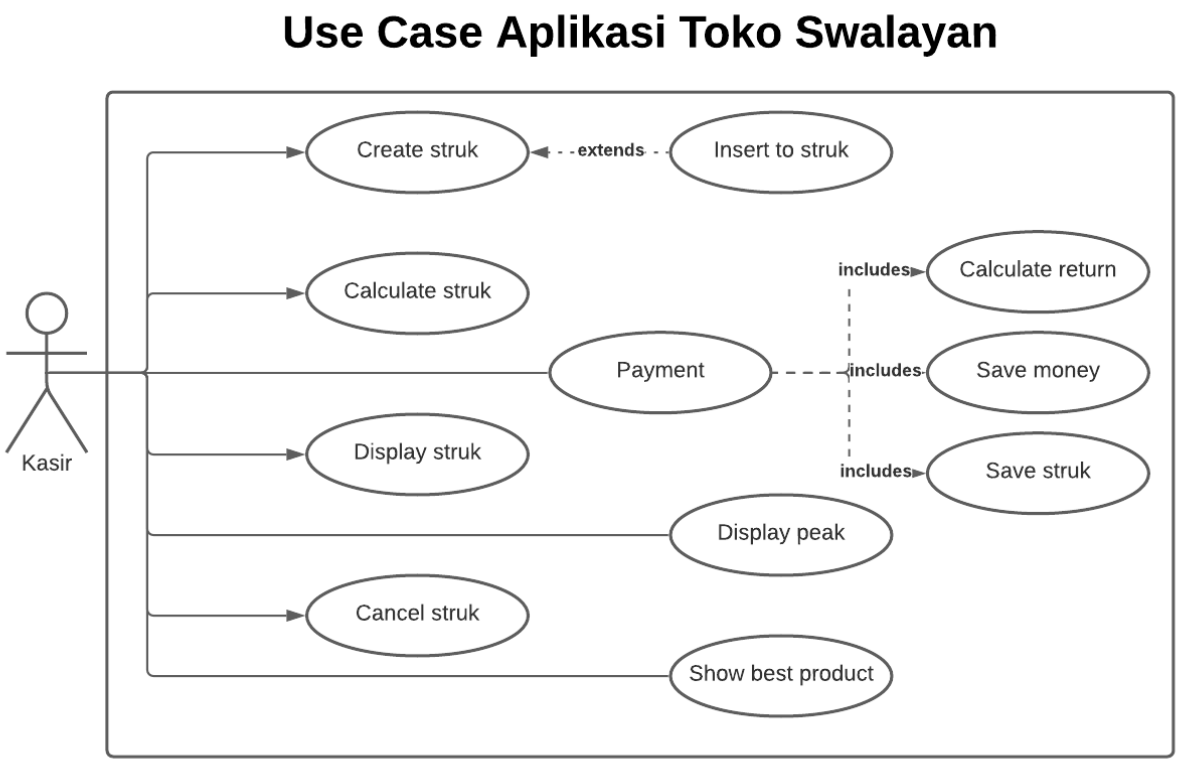
2021

Deskripsi Aplikasi

Toko Swalayan Algoritma dan Pemrograman B adalah aplikasi berbasis Command Line Interface yang dibangun sebagai sistem kasir untuk memfasilitasi kebutuhan pencatatan transaksi dan pengelolaan transaksi. Selain itu, aplikasi ini juga akan digunakan oleh pemilik toko swalayan untuk mendapatkan insight-insight tertentu terkait pendapatan yang diperoleh dalam periode waktu tertentu berdasarkan data transaksi yang dimilikinya.

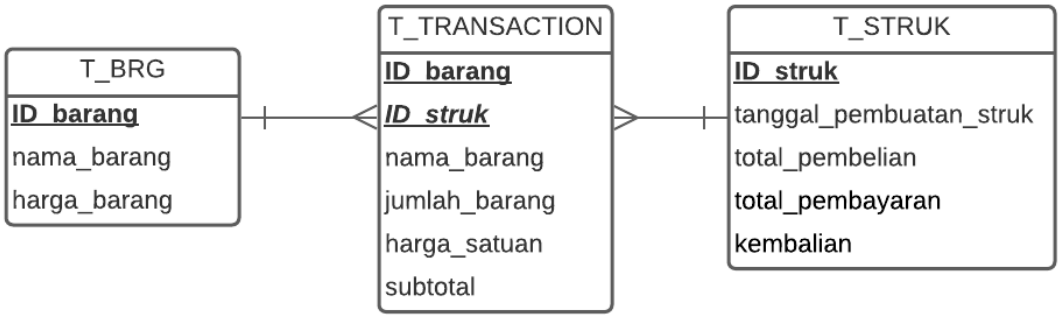
Deskripsi Use Case

Berikut ini adalah use case dari aplikasi Swalayan.



Rancangan Database (ER)

Berikut ini adalah rancangan database (ERD) dari aplikasi kami.



T_BRG = (**ID_barang**, nama_barang, harga_barang)
T_TRANSACTION = (**ID_struk**, **ID_barang**, nama_barang, jumlah_barang, harga_satuan, subtotal)
T_STRUK = (**ID_struk**, tanggal_pembuatan_struk, total_pembelian, total_pembayaran, kembalian)
Foreign Key = T_TRANSACTION(ID_barang) → T_BRG(ID_barang)
= T_TRANSACTION(ID_struk) → T_STRUK(ID_struk)

Contoh Input dan Output

Udaa oke si harusnya use casenya, tinggal ke contoh I/O nya di ss-ssin, kalau ada case yg belum ke cover, dilisting aja apa aja yang belum kecover casenya

Case 1: Create Struk

```
>>> CREATE_STRUK
CREATE_STRUK sukses. ID Struk: 35980. Struk aktif: 35980
```

Case 2: Insert Struk

```
>>> INSERT "Pepsodent" 11
INSERT pada struk 35980 sukses. Barang Pepsodent. Jumlah barang 11.
```

Case 2.1: Insert Struk Barang Tidak Tersedia

```
>>> INSERT "Anak Ayam" 9
INSERT pada struk 35980 gagal. Barang Anak Ayam tidak dikenal.
```

Case 3: Calculate Struk

```
>>> CALCULATE_STRUK
CALCULATE_STRUK pada struk 35980 berhasil. Total pembelian adalah 16500.
```

Case 4: Payment

```
PAYMENT pada struk 35980 berhasil. Pembayaran 100000. Total Pembelian 16500. Kembalian
 83500. Struk berhasil disimpan dan dihapus dari struk aktif.
```

Case 4.1: Payment kurang

```
PAYMENT pada struk 35980 gagal. Pembayaran tidak cukup.
>>> PAYMENT 100000
```

Case 5: Display Struk

```
>>> DISPLAY_STRUK
   ID_struk tanggal_pembuatan_struk  total_pembelian  total_pembayaran  kembalian
0     56789              20-11-2021            30000             50000      20000
1     56790              21-11-2021            30000             60000      30000
2     56791              22-11-2021            30000             70000      40000
3     77957              18-11-2021           180000            200000      20000
4     10566              18-11-2021            30000             50000      20000
5     56119              18-11-2021            25000             50000      25000
6     64304              01-12-2021           840000           1000000     160000
7     35980              02-12-2021            16500            100000      83500
8     57185              02-12-2021           511500           1000000     488500
9     95720              02-12-2021            78000           1000000     922000
10    30143              02-12-2021           421500           1000000     578500
```

Case 5.1: Display Struk Tanggal tertentu

```
>>> DISPLAY_STRUK 18-11-2021 20-11-2021
  ID_struk tanggal_pembuatan_struk  total_pembelian  total_pembayaran  kembalian
0    56789              20-11-2021            30000             50000      20000
3    77957              18-11-2021           180000            200000      20000
4    10566              18-11-2021            30000             50000      20000
5    56119              18-11-2021            25000             50000      25000
```

Case 6: Cancel Struk

```
>>> CREATE_STRUK
CREATE_STRUK sukses. ID Struk: 21595. Struk aktif: 21595
>>> CANCEL_STRUK
STRUK 21595 berhasil dihapus dari memori.
```

Case 6.1: Cancel Struk Saat Tidak Ada Struk Aktif

```
>>> CANCEL_STRUK
CANCEL_STRUK gagal. Tidak ada struk aktif. Silakan membuat struk.
```

Case 8: Display peak

```
>>> DISPLAY_PEAK
                        total_pembayaran  count
tanggal_pembuatan_struk
18-11-2021                        450000      6
01-12-2021                       3000000      3
02-12-2021                        100000      1
20-11-2021                         50000      1
21-11-2021                         60000      1
22-11-2021                         70000      1
```

Case 8.1: Display Peak Tanggal Tertentu

```
>>> DISPLAY_PEAK 18-11-2021 22-11-2021
                        total_pembayaran  count
tanggal_pembuatan_struk
18-11-2021                        450000      6
20-11-2021                         50000      1
21-11-2021                         60000      1
22-11-2021                         70000      1
```

Case 8.2: Display Peak Tanggal Invalid

```
>>> DISPLAY_PEAK 18-11-2021 17-11-2021
Date tidak valid. Gunakan format DD-MM-YYYY dan pastikan end_date > start_date.
```

Case 9: Display Best Product

```
>>> BEST_PRODUCT
        ID_barang  total_pembelian  jumlah_barang         nama_barang  harga_barang
0      CADBURY150          2101500           58.0       Cadbury 150gr         15000
1           PEPSO          1262500           67.0           Pepsodent          1500
2  SILVERQUEEN150          1261500           28.0   Silverqueen 150gr         12000
3            AQUA          1041000          116.0  Aqua Botol 1 Liter          5000
4           MIE_1           133000           27.0      Indomie Goreng          2000
```

Case 9.1: Display Best Product Tanggal Tertentu

```
Date tidak valid. Gunakan format DD-MM-YYYY dan pastikan end_date > start_date.
>>> BEST_PRODUCT 18-11-2021 21-11-2021
  ID_barang  total_pembelian  jumlah_barang         nama_barang  harga_barang
0     PEPSO           235000           32.0           Pepsodent          1500
1     MIE_1            55000           10.0      Indomie Goreng          2000
2      AQUA            30000            1.0  Aqua Botol 1 Liter          5000
```

Case 9.1: Display Best Product Tanggal Invalid

```
>>> BEST_PRODUCT 18-11-2021 17-10-2021
Date tidak valid. Gunakan format DD-MM-YYYY dan pastikan end_date > start_date.
```

Case 10: End to End Purchase Process

```
>>> CREATE_STRUK
CREATE_STRUK sukses. ID Struk: 30143. Struk aktif: 30143
>>> INSERT "Aqua Botol 1 Liter" 9
INSERT pada struk 30143 sukses. Barang Aqua Botol 1 Liter. Jumlah barang 9.
>>> INSERT "Pepsodent" 7
INSERT pada struk 30143 sukses. Barang Pepsodent. Jumlah barang 7.
>>> INSERT "Cadbury 150gr" 18
INSERT pada struk 30143 sukses. Barang Cadbury 150gr. Jumlah barang 18.
>>> INSERT "Silverqueen 150gr" 8
INSERT pada struk 30143 sukses. Barang Silverqueen 150gr. Jumlah barang 8.
>>> CALCULATE_STRUK
CALCULATE_STRUK pada struk 30143 berhasil. Total pembelian adalah 421500.
>>> PAYMENT 1000000
PAYMENT pada struk 30143 berhasil. Pembayaran 1000000. Total Pembelian 421500. Kembalian
578500. Struk berhasil disimpan dan dihapus dari struk aktif.
```

Modul dan Kode Program

Struktur & Modul Program

Berikut ini adalah struktur kode program secara umum.

```
+---components
|   +---_init_.py
|   +---Database.py
|   +---Struk.py
|   +---System.py
|   \---Transaksi.py
|
+---database
|   +---T_BRG.csv
|   +---T_STRUK.csv
|   \---T_TRANSAKSI.csv
|
\---app.py
```

Keterangan:

1. components: berisi modul-modul yang diperlukan untuk aplikasi Swalayan (dikemas dalam pendekatan *object-oriented programming*)
    a. Database.py : modul pengelolaan basis data
    b. Struk.py : modul pengelolaan struk yang dikelola di aplikasi
    c. System.py : modul pemrosesan input command dari user
    d. Transaksi.py : modul pengelolaan transaksi dari struk
2. database: tempat penyimpanan data-data dalam pengelolaan aplikasi swalayan.
    a. T_BRG.csv : menyimpan informasi barang yang dijual
    b. T_STRUK.csv : menyimpan struk yang dihasilkan
    c. T_TRANSAKSI.csv : menyimpan rincian transaksi yang terjadi
3. app.py: aplikasi utama (*main program*) untuk menjalankan aplikasi swalayan.

Kode Program

Berikut ini adalah kode program yang digunakan dalam aplikasi:

Prerequisites:

- Well documented program
- Passed QA

1. Database.py

```
import pandas as pd
from datetime import datetime

PATH = "database/"

class Database:
    '''
    Class Database to provide database CRUD functionality on System
    Using CSV Format as an implementation of Database
    '''

    def __init__(self):
        '''
        Construct new Database Object
        '''
        self.tabel_barang = pd.read_csv(PATH + "T_BRG.csv")
        self.tabel_barang['ID_barang'] =
self.tabel_barang['ID_barang'].astype(str)
        self.tabel_struk = pd.read_csv(PATH + "T_STRUK.csv")
        self.tabel_struk['total_pembayaran'] =
self.tabel_struk['total_pembayaran'].astype(int)
        self.tabel_transaksi = pd.read_csv(PATH + "T_TRANSAKSI.csv")

    def insert_struk(self, data):
        '''
        Insert the struk into the tabel_struk
        '''
        self.tabel_struk = self.tabel_struk.append(data, ignore_index=True)

    def insert_transaksi(self, data):
        '''
        Insert the new transaction into the tabel_transaksi
        '''
        self.tabel_transaksi = self.tabel_transaksi.append(data,
ignore_index=True)

    def select_barang(self, barang):
        '''
        Return a row with criteria such as nama_barang equals to barang
        '''
        return self.tabel_barang[self.tabel_barang['nama_barang'] ==
barang].iloc[0]

    def _convert_date(self, date):
        return datetime.strptime(date, '%d-%m-%Y')

    def select_struk(self, start_date="", end_date=""):
```

```
        '''
        Select all struk between start_date and end_date
        If the start_date and end_date is None, return all struk all time
        If the end_date is None, then return all struk since start_date
        '''
        result = self.tabel_struk
        if (start_date):
          result =
result[result['tanggal_pembuatan_struk'].map(self._convert_date) >=
self._convert_date(start_date)]
        if (end_date):
          result =
result[result['tanggal_pembuatan_struk'].map(self._convert_date) <=
self._convert_date(end_date)]
        return result

    def select_peak(self, start_date="", end_date=""):
        '''
        Return all top 10 peak sales between start_date and end_date
        If the start_date and end_date is None, return top 10 peak sales all time
        If the end_date is None, then return top 10 peak sales since start_date
        '''
        result = self.select_struk(start_date=start_date, end_date=end_date)
        joined_result = result.merge(self.tabel_transaksi[['ID_struk',
'ID_barang']], on='ID_struk', how='left')
        joined_result['count'] =
joined_result.groupby('tanggal_pembuatan_struk')['tanggal_pembuatan_struk'].trans
form('count')
        joined_result =
joined_result.groupby('tanggal_pembuatan_struk').aggregate({'total_pembayaran':'s
um', 'count': 'max'}).sort_values('count', ascending=False)
        return joined_result[:10]

    def select_best_product(self, start_date="", end_date=""):
        '''
        Return all top 5 best product between start_date and end_date
        If the start_date and end_date is None, return top 5 best product all
time
        If the end_date is None, return top 5 best product since start_date
        '''
        result = self.select_struk(start_date=start_date, end_date=end_date)
        joined_result = result.merge(self.tabel_transaksi[['ID_struk',
'ID_barang', 'jumlah_barang']], on='ID_struk',
how='left').groupby('ID_barang').aggregate({'total_pembelian': 'sum',
'jumlah_barang': 'sum'}).merge(self.tabel_barang, on='ID_barang',
how='left').sort_values(by='total_pembelian', ascending=False)
        joined_result = joined_result.reset_index(drop=True)
        return joined_result[:5]

    def save(self):
        '''
        Save the updated database into the csv
        '''
        self.tabel_transaksi.to_csv(PATH + "T_TRANSAKSI.csv", index=False)
        self.tabel_struk.to_csv(PATH + 'T_STRUK.csv', index=False)

DB = Database()
```

2. Struk.py

```
from datetime import date
import random

from .Transaksi import Transaksi
from .Database import DB

class Struk:
    '''
    Class Struk to represent Struk in Toko Swalayan System
    '''

    def __init__(self):
        '''
        Constructor for Struk
        '''
        self.id = str(random.randint(10000, 99999))
        self.date = date.today().strftime("%d-%m-%Y")
        self.total = 0
        self.payment = 0
        self.exchange = 0
        self.transactions = []

    def insert(self, barang, jumlah):
        '''
        Insert new transaction into the struk
        Raise an error if such item doesn't exist on the database
        '''
        try:
            transaksi = Transaksi(barang, jumlah, self.id)
            self.transactions.append(transaksi)
        except:
            raise Exception("INSERT pada struk %s gagal. Barang %s tidak dikenal.
" % (self.id, barang))

    def calculate(self):
        '''
        Calculate and return the total of all transaction subtotal
        '''
        self.total = 0
        for transaction in self.transactions:
            self.total += transaction.subtotal
        return self.total

    def to_csv(self):
        '''
        Return CSV Formatted for Struk
        '''
        return {
            "ID_struk": self.id,
            "tanggal_pembuatan_struk": self.date,
            "total_pembelian": self.total,
            "total_pembayaran": self.payment,
            "kembalian": self.exchange,
        }

    def set_payment(self, payment):
        '''
        Doing the payment based on given payment nominal
```

```
        If the payment nominal is below the total nominal, raise an error
        If the struk has not be calculated, raise an error
        '''
        if (self.total == 0):
            raise Exception('PAYMENT pada struk ' + self.id + ' gagal. Anda belum
melakukan kalkulasi atau menentukan barang yang dibeli.')

        elif (payment < self.total):
            raise Exception('PAYMENT pada struk ' + self.id + ' gagal. Pembayaran
tidak cukup.')

        else:
            self.payment = payment
            self.exchange = payment - self.total
            for transaction in self.transactions:
                DB.insert_transaksi(transaction.to_csv())
            DB.insert_struk(self.to_csv())
            DB.save()
```

3. System.py

```
from .Struk import Struk
from .Database import *
import re
from datetime import datetime

# System Class
class System:
    '''
    Class System acts as Swalayan System Representation
    '''

    def __init__(self):
        '''
        System Constructor
        '''
        self.active_struk = None

    def run(self, command: str):
        '''
        Run the user command and evaluate it
        If the command is invalid or there is an error on system, print the error
message
        '''
        try:
            command_list = command.split(" ", 1)
            main_command = command_list[0]
            rest_command = command_list[1] if (len(command_list) == 2) else None
            if (main_command == 'CREATE_STRUK'):
                self.create_struk()
            elif (main_command == 'INSERT'):
                self.insert_struk(rest_command)
            elif (main_command == 'CALCULATE_STRUK'):
                self.calculate_struk()
            elif (main_command == 'PAYMENT'):
                self.payment(rest_command)
            elif (main_command == 'CANCEL_STRUK'):
                self.cancel_struk()
```

```
            elif (main_command == 'DISPLAY_STRUK'):
                self.display_struk(rest_command)
            elif (main_command == 'DISPLAY_PEAK'):
                self.display_peak(rest_command)
            elif (main_command == 'BEST_PRODUCT'):
                self.best_product(rest_command)
            elif (main_command == 'EXIT'):
                quit()
            elif (main_command == 'HELP'):
                self.help()
            else:
                raise Exception("Command tidak valid. Gunakan HELP untuk melihat
cara penggunaannya.")

        except Exception as e:
            print(e)

    def create_struk(self):
        '''
        Create Struk functionality for create new struk
        '''
        self.active_struk = Struk()
        print("CREATE_STRUK sukses. ID Struk: %s. Struk aktif: %s" %
(self.active_struk.id, self.active_struk.id))

    def insert_struk(self, rest_command):
        '''
        Insert new transaction into the active struk
        '''
        self.check_active_struk(error_message="INSERT gagal. ")
        is_valid = re.match("\"[\w\s]+\" [0-9]+", rest_command)
        if (is_valid):
            rest_command_arr = rest_command.rsplit(" ", 1)
            nama_barang, jumlah_barang = rest_command_arr[0].replace('"', ''),
int(rest_command_arr[1])
            self.active_struk.insert(nama_barang, jumlah_barang)
            print("INSERT pada struk %s sukses. Barang %s. Jumlah barang %d." %
(self.active_struk.id, nama_barang, jumlah_barang))
        else:
            raise Exception("INSERT pada struk %s gagal. Sintaks salah." %
(self.active_struk.id))

    def calculate_struk(self):
        '''
        Calculate Struk functionality
        '''
        self.check_active_struk(error_message="CALCULATE_STRUK gagal. ")
        self.active_struk.calculate()
        print("CALCULATE_STRUK pada struk %s berhasil. Total pembelian adalah
%d." % (self.active_struk.id, self.active_struk.total))

    def payment(self, rest_command):
        '''
        Doing the payment for the active struk
        '''
        self.check_active_struk(error_message="PAYMENT gagal. ")
        nominal = int(rest_command)
        self.active_struk.set_payment(nominal)
        print('PAYMENT pada struk ' + self.active_struk.id + ' berhasil. '\
            'Pembayaran ' + str(self.active_struk.payment) + '. Total Pembelian '
```

```
+ str(self.active_struk.total) + '. '\
            'Kembalian ' + str(self.active_struk.exchange) + '. Struk berhasil
disimpan dan dihapus dari struk aktif.')
        self.active_struk = None

    def cancel_struk(self):
        '''
        Delete the active struk
        '''
        self.check_active_struk(error_message="CANCEL_STRUK gagal. ")
        print("STRUK %s berhasil dihapus dari memori." % self.active_struk.id)
        self.active_struk = None

    def _validate_range_date(self, start_date, end_date):
        if (len(start_date) == 0 or len(end_date) == 0):
            return True
        start = datetime.strptime(start_date, '%d-%m-%Y')
        end = datetime.strptime(end_date, '%d-%m-%Y')
        return start <= end

    def _validate_date(self, date):
        if (len(date) == 0):
            return True
        datetime.strptime(date, '%d-%m-%Y')
        return True

    def _extract_range_time(self, rest_command=''):
        '''
        Extract range time from rest_command and return start_date and end_date
        '''
        if (rest_command == None):
          rest_command = ''
        start_date = ''
        end_date = ''
        rest_command_arr = rest_command.split(" ")
        try:
          start_date = rest_command_arr[0]
          end_date = rest_command_arr[1]
        except:
          pass
        try:
          self._validate_date(start_date)
          self._validate_date(end_date)
          if (not self._validate_range_date(start_date, end_date)):
              raise Exception()
        except:
          raise Exception("Date tidak valid. Gunakan format DD-MM-YYYY dan
pastikan end_date >= start_date.")
        return start_date, end_date

    def display_struk(self, rest_command):
        '''
        Display Struk Functionality to return all struk based on user command
        '''
        start_date, end_date = self._extract_range_time(rest_command)
        result = DB.select_struk(start_date=start_date, end_date=end_date)
        print(result)

    def display_peak(self, rest_command):
        '''
```

```
        Display Peak Functionality to return all peak date for the sales based on
user command
        '''
        start_date, end_date = self._extract_range_time(rest_command)
        result = DB.select_peak(start_date=start_date, end_date=end_date)
        print(result)

    def best_product(self, rest_command):
        '''
        Best Product Functionality to return all best product based on user
command
        '''
        start_date, end_date = self._extract_range_time(rest_command)
        result = DB.select_best_product(start_date=start_date, end_date=end_date)
        print(result)

    def check_active_struk(self, error_message):
        '''
        Check if there is an active struk or not
        If there is no active struk, raise an error
        '''
        if (self.active_struk == None):
            raise Exception(error_message + "Tidak ada struk aktif. Silakan
membuat struk.")

    def help(self):
        '''
        Show the help information about how to use the system and accepted
command
        '''
        help_info = "ACCEPTED COMMAND: \n\
            1. CREATE_STRUK \n\
            2. INSERT <nama_barang> <jumlah_barang> \n\
            3. CALCULATE_STRUK \n\
            4. PAYMENT <nominal> \n\
            5. CANCEL_STRUK \n\
            6. DISPLAY_STRUK <tanggal_awal> <tanggal_akhir> \n\
            7. DISPLAY_PEAK <tanggal_awal> <tanggal_akhir> \n\
            8. BEST_PRODUCT <tanggal_awal> <tanggal_akhir> \n\
            9. HELP \n\
            10. EXIT \n"

        print(help_info)
```

4. Transaksi.py

```
from .Database import *

class Transaksi:
    '''
    Class Transaksi to store transaction item of the struk
    '''

    def __init__(self, barang, jumlah, id_struk):
        '''
        Construct new transaksi item
        '''
        data = DB.select_barang(barang)
```

```
        self.id_barang = data['ID_barang']
        self.id_struk = id_struk
        self.barang = barang
        self.jumlah = jumlah
        self.harga_satuan = data['harga_barang']
        self.subtotal = self.jumlah * self.harga_satuan

    def to_csv(self):
        '''
        Return CSV formatted data of the transaksi
        '''
        return {
            "ID_struk": self.id_struk,
            "ID_barang": self.id_barang,
            "nama_barang": self.barang,
            "jumlah_barang": self.jumlah,
            "harga_satuan": self.harga_satuan,
            "subtotal": self.subtotal,
        }
```

5. app.py

```
from components.System import *

System = System()
print("Selamat datang di Toko Swalayan Algoritma dan Pemrograman B!")
command = input(">>> ")
while (command != "EXIT"):
    System.run(command)
    command = input(">>> ")
print("Terima kasih telah menggunakan kasir Algopro!")
```