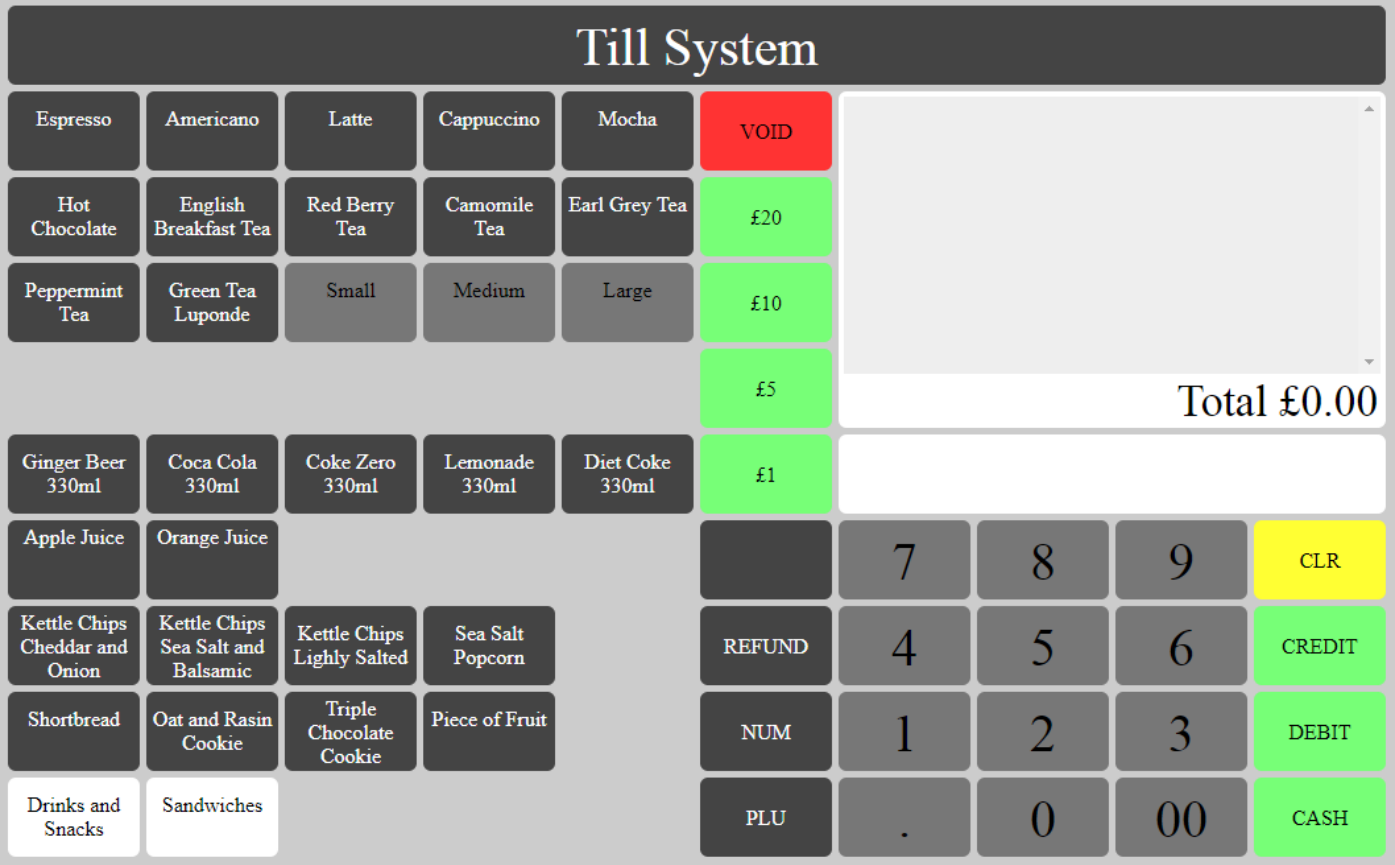
CM50267 Software Technologies for Data Science

Mini-Project: Point of Sale System

Specification



Deadline: 14th December 2018, 5pm

Introduction

In this coursework you will undertake two tasks.

1. The first is to implement the backend server for a simple point of sale (POS) system.

2. The second is to perform analysis of the data collected by such a system.

You will be provided with:

1. A complete browser based front-end for the POS till.

2. A skeleton backend framework for the POS.

3. A sqlite3 database containing the product table.

4. A csv file containing a series of till transactions for you to analysis in addition to those you generate yourself.

Part One

The till can generate the following action requests to the server:

1. action=plu

   The following additional parameters will be set:
   refund = 4 if the REFUND key was selected.
   refund = 0 if the REFUND key was not selected.
   num=1 or the number entered preceding the selection of the NUM key.
   plu=… the value entered into the input box prior to selection of the PLU key or pressing enter

   When received the server should lookup the plu code in the products table and if it exists add this item to the current transaction. Refund and num indicate how many of the item should be included and if this is a sale or refund.

   If there is no current transaction, a new transaction should be created and become the current transaction.

2. action=cash

   The following additional parameters will be set:
   refund = 4 if the REFUND key was selected.
   refund = 0 if the REFUND key was not selected.
   type=1,2, or 3 indicating Cash, Debit or Credit.
   cash=… the value entered into the input box prior to selection of the CASH, DEBIT or CREDIT key.

   This action indicates a payment (or refund was made). The payment should be recorded against the current transaction. If the payment matches the current outstanding balance for the transaction, the transaction should be completed and closed. If the payment exceed the outstanding balance for the transaction, an additional cash payment refund should be recorded with the transaction to bring the balance to zero (the change) and the transaction closed.

3. action=program

   The following additional parameters will be set:
   refund = 4 if the REFUND key was selected.
   refund = 0 if the REFUND key was not selected.
   num=1 or the number entered preceding the selection of the NUM key.
   pos= the value assigned to the programmable key that generated this request.
   shift= 0 or a shift value as a result of as shift key being pressed.
   value=… the value entered into the input box prior to selection of the programmable key.

   When received the server should lookup the products table to find the product that matches the pos and shift values and if it exists add this item to the current transaction. Refund and num indicate how many of the item should be included and if this is a sale or refund.

   If there is no current transaction, a new transaction should be created and become the current transaction.

   If the shift value supplied is zero, it can match the entry in the table with a shift of zero or any negative shift.

   If the shift value supplied is non zero, it can match the entry in the table with a matching shift value or matching negative valued shift.

4. action=clr

   There are no additional parameters. This request is generated when the void key is selected. If a transaction is active, it should be voided in its entirety.

5. action=status

   There are no additional parameters. This request is generated when the till page is loaded or one of it's alternative views is loaded. If there is a currently active transaction the system should provide a response that allows the display to be updated to reflect this.

On completion of the action request, the server must provide an XML formatted response that may contains the following <action>…</action> entries.

<action>
<type>reset</type>
</action>

This causes the internal state of the frontend to be reset on the next action it detects, before it responds to the action. It should be included as the last action in a server response when a transaction is closed.

<action>
<type>total</type>
</value>…</value>
</action>

The value field should contain an integer number that indicates the value in pence to be displayed in the onscreen Total field.

<action>
<type>refill</type>
<where>…</where>
<what>…</what>
</action>

Refill allows the server to supply html code that should be placed in either the on screen receipt box or the title bar. The title is identified using <where>title</where> and the receipt box is identified using <where>target</where>.

The content should be base64 encoded to avoid character escaping issues and occupies the <what>…</what> field.

<action>
<type>append</type>
<where>…</where>
<what>…</what>
</action>

Append is identical to refill but it concatenates the supplied content with the existing content of the .

The skeleton server demonstrates the use of these by updating the total field and placing a 'Change…' message in the title bar and the parameters of the request to which it is responding in the receipt box.

Part One

Your task is to develop the skeleton server into a functional till system. You can assume that only one browser session will access the server. You should run the server by executing py till.py from the command prompt while in the directory containing the support files. Start a Chome web browser and access http://localhost

Task 1.1 (2 Marks)

You are provided with a basic python based backend server. Unfortunately it's rather old code and need a little updating. It was written in the days of python2. You should spend a little time updating it for python3.

Task 1.2 (8 Marks)

Add support to the back-end python server for each of the incoming actions types that the front-end may deliver to the backend. You should validate the input parameters and protect against malicious requests. You should generate suitable XML responses and update the database as appropriate.

You should not change the frontend. You can create whatever database tables you require, including modifying the supplied tables. At a minimum you will require a transaction table that records each separate transaction, this is defined as everything that appears on a single receipt. You will also need an item table that records the details of each item that is part of the transaction.

The progress of each transaction should appear in the receipt box on screen. You will need to apply some basic HTML formatting to the output.

Task 1.3 (10 Marks)

Add support to automatically calculate the following offers.

1. The Meal Deal: A free packet of crisps, popcorn or piece of fruit when a sandwich/panini and a drink are purchased.

2. Hot Drink and a Cookie: A cookie or shortbread half price when a hot drink is purchased.

All deals that apply should be detected automatically and applied immediately so that they are reflected in the total shown as the transaction progresses. Each application of a deal should be included as an item for that transaction with the value indicating the discount applied. Only one deal should be applied to any one item. E.g. A purchased hot drink can be part of the meal deal or the cookie offer but not both. However if two drinks were purchased one could be used as part of the meal deal and the other as part of the cookie offer.

Your approach should generalise to allow similar deals with different products without having to change the server code, only the database.

Part Two

Task 2.1 (6 Marks)

Create a separate python program that is able to access the database and provide the following reports in CSV format.

1. A daily sales report that shows how much of each product has been sold, and the total value of sales. It should also show the total cash, debit and card payments taken. It should also lists all discounts given, both the number and value.

2. A product report that, for a given product shows the total volume and value of sales for a product on each day during the reporting period.

3. Export all of the item data including the transaction to which it belong to a csv file for use in Task 2.2

Task 2.2 (14 Marks)

Write a further python program that given a csv file containing the details of all the transactions applies the Apriori Algorithm to find groupings of products that are frequently sold together to assist the café owner in deciding how to best layout the products or decide on future offers.

The blog at http://blog.hackerearth.com/beginners-tutorial-apriori-algorithm-data-mining-r-implementation provides a good overview of the algorithm. And includes the Walmart Beer and Nappies story with which all data scientists should familiar.

Deliverables

You should provide the following by the submission deadline.

1. Your updated code for the backend-server.
2. The code you created for tasks 2.1 and 2.2.
3. Screenshots showing the progress of a series of transactions.
4. An example of each type of report from Task 2.1
5. The results of running your Task 2.2 code on the transactions you have gathered during development.
6. The results of running your Task 2.2 code on a supplied csv file.
7. An implementation report that is intended to act as a reminder of what you have done to assists you should you return to the code at a future date.

Supplied Files and Data

The following files are provided:

till.html	The frontend main page (html). You should not need to modify this.
till2.html	The frontend second page (html). You should not need to modify this.
404.html	The frontend error page (html). You should not need to modify this.
till.css	The frontend cascading style sheet for the html pages (css). You should not need to modify this.
till.js	The frontend javascript code that provided the till interface functionality, makes AJAX requests to the server and process the responses. You should not need to modify this.
till.py	The backend python server framework code.
products.sql	The code to define the basic set to products that match those presented in the till.html and till2.html pages.
dataset.csv	A larger dataset for Task 2.2.