



spend & track

USER GUIDE

AB0403 SEMINAR GROUP 12: TEAM 6

DR. YEO WEE KIANG

Ariella	U1910990L
Foo Jun Kit	U1910482K
Pua Tian Yu	U1910659B
Teo Yin Xin, Brenda	U1910448H

Table of Contents

Work Distribution	3
Project Objectives	4
Prototype Features & Functionalities	5
Set-Up	5
Account Creation and Login	6
Budget Viewing	8
Budget Editing	10
Keying in New Expense Entry	12
Editing Existing Expense Entry	14
Exporting Data (CSV)	16
User Manual	21
[1] Logging In:	21
[2] View Budget:	23
[3] Edit Budget:	24
[4] Entering a new expense entry:	26
[5] Editing an existing expense entry:	29
[6] Exporting data into a CSV file:	31
[7] Exit Programme:	33

Work Distribution

Our team conducted a series of virtual meetings to develop the code for Spend & Track together. This was decided to be the most efficient method, as it would ensure that every member was kept up to date with the progress of the code.

During the first meeting, the key features which were desired to be included in Spend & Track were discussed and decided upon. This established a good direction for the business expense prototype. Thereafter, our team assigned different work responsibilities to each member based on these features. The assignment was broadly as follows:

Feature / Work Responsibility	Member(s)-In-Charge
Account Creation and Login	Jun Kit, Tian Yu
Budget Viewing	Brenda
Budget Editing	Tian Yu
Keying in New Expense Entry	Jun Kit
Editing Existing Expense Entry	Ariella, Brenda
Exporting Data (CSV)	Ariella

Despite the division of workload based on the features of the prototype, our team has also agreed to formulate the codes for the features that we were not individually assigned to. This is so that the team will be able to learn from one another, suggest ways to improve the codes and to facilitate the streamlining of the codes for each feature.

After working on the individual features, our team came together to join the pieces of code. This took several virtual meetings but was a smooth process given that every member contributed to the best of their abilities.

Once the code and flow has been set up, each member took the time to run through each option in each feature to check for any errors. This process of trial and error done by all the members was beneficial as it uncovered parts of the code which did not work or produced errors. Our team then worked together to address these issues and fix these errors. This resulted in a robust and workable expense tracker.

Project Objectives

Tracking of expenses is integral to managing finances, not just for individuals, but for businesses as well. For businesses, expenses are usually incurred on a daily basis, giving rise to the need to track expenses at an early stage. This helps to ensure that expenses records are kept well-organised to facilitate routine business processes such as budget planning, tax filing and to deter employees or individuals from spending beyond the allocated or set budget for their department.

Leveraging on this need, our team's objective is to design a simple yet secure expense tracker prototype to aid businesses in managing their costs in order to improve their profitability. Spend & Track serves to provide users with the necessary functionalities of an expense tracker such as the adding and editing of monthly expenses and budgets. These features are further enhanced with other features such as access restrictions depending on the employee's position and department, providing user alerts if expense entry is about to exceed the set departmental budget as well as the generation and exporting of monthly departmental expense reports.

Prototype Features & Functionalities

Set-Up

The organisation used in our team's dataset comprises four departments and four possible employee positions. The budget for each department is generally fixed for every month at \$5,000, \$10,000, \$2,000 and \$2,000 for the Finance, Marketing, Sales and Human Resource department respectively. However, the monthly budget for each department can be further edited using the Budget Editing feature. The monthly budget for each department is stored in a dictionary, with the department name as the key and the corresponding departmental budget as the value.

```
### Database Set Up

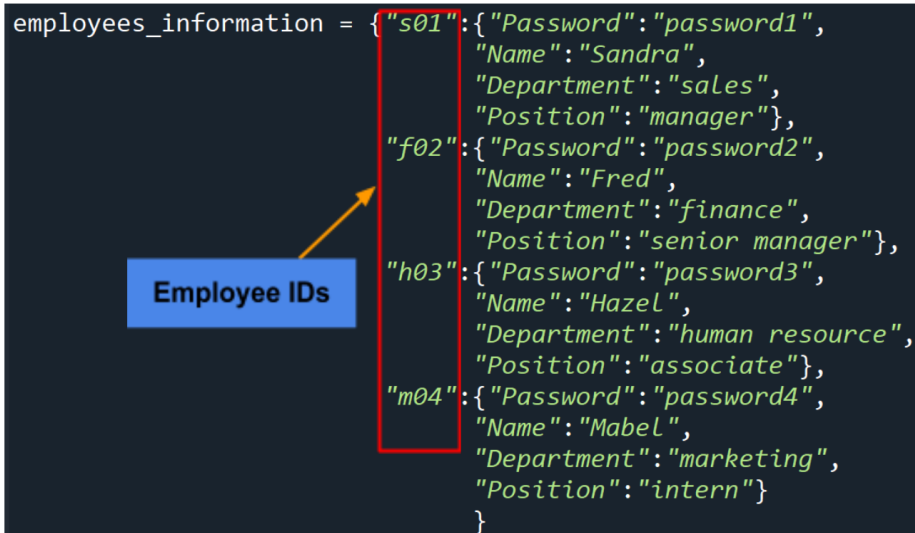
positions = ["intern", "associate", "manager", "senior manager"]
departments = ["finance", "marketing", "sales", "human resource"]
expense_categories = ["Stationery", "Salary", "Marketing", "Miscellaneous", "Others"]
months = ["JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"]

departmental_monthly_budget = {"finance":5000, "marketing":10000, "sales":2000, "human resource":2000}
```

Similarly, the beginning and remaining monthly departmental budget are also stored in a dictionary which will be subsequently used in Spend & Track's features such as the viewing and editing of budget for a specific department during a specific month.

```
departmental_budget_balance = {}
for month in months:
    departmental_budget_balance[month] = {}
    for dept, budg in departmental_monthly_budget.items():
        departmental_budget_balance[month][dept] = {"Beginning":budg, "Remaining":budg}
```

Our team has also formulated a predefined set of employee information and stored it in a dictionary. Each employee and their corresponding information such as password, name, department and position are identified based on their employee ID (e.g. "s01", "f02", "h03" and "m04"). The alphabet in the employee ID represents the department that the employee belongs to (e.g. s for Sales department, f for Finance department etc.). These employee IDs will form the keys and the employee information will form the values in the dictionary.



```
employees_information = {"s01": {"Password": "password1",
                                "Name": "Sandra",
                                "Department": "sales",
                                "Position": "manager"},
                        "f02": {"Password": "password2",
                                "Name": "Fred",
                                "Department": "finance",
                                "Position": "senior manager"},
                        "h03": {"Password": "password3",
                                "Name": "Hazel",
                                "Department": "human resource",
                                "Position": "associate"},
                        "m04": {"Password": "password4",
                                "Name": "Mabel",
                                "Department": "marketing",
                                "Position": "intern"}
}
```

Account Creation and Login

Prior to obtaining access to the main menu of Spend & Track, employees will be prompted to indicate whether they are a new or an existing user. Employees that do not have an existing account will be led through a series of prompts to input the necessary information required to set up an account such as employee ID, password, position and department.

After a new account has been successfully created, an employee record summary will be reflected before the main menu is displayed. On other hand, employees with an existing account will immediately be taken to the main menu of Spend & Track upon successful entry of their employee ID and password.

Based on the employee ID input, Spend & Track will also perform a check in their existing account database to ensure that no existing account is linked to the employee ID through the usage of nested if statements.

Should the user indicate that they are a new user, yet an existing account is found based on their employee ID, the user will be prompted to enter their password. If the password entered is correct, an employee record summary will be shown as a refresher and the user will be taken to the Spend & Track main menu.

```

employee_input = input("Welcome! Are you a new user? Y/N ").upper()
# NEW USER
while employee_input not in ["Y", "N"]:
    employee_input = input("Invalid response received. Please try again.\n").upper()

if employee_input.upper() == "Y":
    employee_ID_new = input("Please enter your new employee ID: ").lower()

    # Check if is actually existing user
    if employee_ID_new in employees_information.keys():
        print("You are an existing user!")
        password_check()
    else:
        password_creator()
        get_name()
        get_position()
        get_department()

```

```

Welcome! Are you a new user? Y/N Y

Please enter your new employee ID: s01
You are an existing user!

Please enter your password: password1
Password is correct
=====
Employee Record Summary:
=====
Employee ID: s01
Password: password1
Employee Name: Sandra
Employee Position: manager
Employee Department: sales
-----
Welcome, Sandra!

What would you like to do today?
[1] View budget
[2] Key in an expense entry
[3] Edit budget
[4] Edit existing expense entry
[5] Export data
[6] End programme
Selection: |

```

Conversely, if a new user indicates that they have an existing account in Spend & Track and enters an employee ID that is not linked to any existing accounts, they will be taken through the account creation process before they can access the menu.

```

else:
    employee_ID_new = input("Please enter your employee ID: ").lower()
    if employee_ID_new in employees_information.keys():
        password_check()
    else:
        print("Sorry! Your ID does not exist. Please create a new account.")
        password_creator()
        get_name()
        get_position()
        get_department()

```

During the account creation or login process, should the user enter an invalid input (e.g. incorrect password, department or invalid position), the system will prompt the user to re-enter their inputs through the use of while loops. In addition, to ensure the strength of passwords, our team has formulated a condition where all passwords must have a minimum of eight characters.

```
def password_creator():
    employees_information[employee_ID_new] = {}
    employee_password_new = input("Please enter your new password: ")
    while len(employee_password_new) < 8:
        print("Password should have at least 8 characters.")
        employee_password_new = input("Please enter your new password: ")
    employee_password_repeat = input("Please confirm your new password: ")
    while employee_password_new != employee_password_repeat:
        employee_password_repeat = input("Please re-confirm your new password: ")
    employees_information[employee_ID_new]["Password"] = employee_password_new
    print("Password created successfully!")
    return

def get_name():
    employee_name_new = input("What is your name? ")
    while employee_name_new.isalpha() is False:
        print("Only alphabets are allowed.")
        employee_name_new = input("What is your name? ")
    employees_information[employee_ID_new]["Name"] = employee_name_new
    return
```

```
Please enter your new password: 111
Password should have at least 8 characters.

Please enter your new password: password5

Please confirm your new password: 111

Please re-confirm your new password: password5
Password created successfully!

What is your name? 1234
Only alphabets are allowed.

What is your name? Jane

Please enter your position (Intern/Associate/Manager/Senior Manager) CEO

You have entered an invalid position. Please enter your position again (Intern/
Associate/Manager/Senior Manager) Intern
```

Budget Viewing

The user can view a simple monthly budget report using the Spend & Track. This feature is defined in a function called `view_budget()`.

```
def view_budget():
    viewing_month = input("Please enter the month you would like to view in MMM format: ").upper()
    while viewing_month not in months:
```

Determining Authority of User

The departmental budgets which are displayed will depend on the department the employee belongs to. For employees working in the *finance department*, they have the authority to view the monthly budget for all departments. For employees working in other departments, they only have the authority to view the monthly budget for the department they belong to.

Such an authorization process is done via a set of “if-else” statements. The employee’s department will automatically be retrieved from the employee information dictionary by using the employee ID (*employee_ID_new*) as the key, which returns a dictionary containing the various pieces of information about the employee. The string “Department” is then passed as the key in this nested dictionary to ultimately return the department of the user.

```
if employees_information[employee_ID_new]["Department"] == "finance":
    print("-"*30)
    print(f"BUDGET RECORDS ({viewing_month.upper()})")

else:
    department_selection = employees_information[employee_ID_new]["Department"]
    amount_selection = departmental_monthly_budget[employees_information[employee_ID_new][department_selection][viewing_month]]
```

Displaying Budget Report (for Finance Employees)

If the department of the user is “finance”, then the budget report for all departments will be displayed. This includes both the beginning budget and the remaining budget. This is done by iterating through the key-value pairs of the *departmental_budget_balance* dictionary belonging to the desired month, and storing them into two variables called *key* and *value*. The department name will be assigned to a variable called *keys*, and the dictionary containing the beginning and remaining budget balance will be assigned to a variable called *value*. The corresponding values can then be displayed as output for each of the departments.

```
for key, value in departmental_budget_balance[viewing_month].items():
    print("Beginning Budget ({}): ${:>10,.2f}".format(key.upper(),value["Beginning"]))
    print("Remaining Budget ({}): ${:>10,.2f}".format(key.upper(),value["Remaining"]))
```

The first print function displays the beginning budget, while the second print function displays the remaining budget.

```
-----
BUDGET RECORDS (OCT)
Beginning Budget (FINANCE): $  5,000.00
Remaining Budget (FINANCE): $  5,000.00
Beginning Budget (MARKETING): $ 10,000.00
Remaining Budget (MARKETING): $ 10,000.00
Beginning Budget (SALES): $   2,000.00
Remaining Budget (SALES): $   2,000.00
Beginning Budget (HUMAN RESOURCE): $  2,000.00
Remaining Budget (HUMAN RESOURCE): $  2,000.00
-----
```

Displaying Budget Report (for Non-Finance Employees)

If the department of the user is not “*finance*” (i.e. sales, marketing or human resource), then the employee can only view the budget report for their own department. For example, an employee in the sales department can only view the budget report for the sales department.

The user’s department name (which was previously identified) will first be assigned to a variable called *department_selection*. Next, the beginning budget is assigned to a variable called *amount_beginning*, while the remaining budget is assigned to a variable called *amount_remaining*.

```
else:
    department_selection = employees_information[employee_ID_new]["Department"]
    amount_beginning = departmental_budget_balance[viewing_month][department_selection]["Beginning"]
    amount_remaining = departmental_budget_balance[viewing_month][department_selection]["Remaining"]
    print("-"*30)
    print(f"BUDGET RECORDS ({viewing_month.upper()})")
    print(f"Beginning Budget ({department_selection.upper()}) : ${amount_beginning:>10,.2f}")
    print(f"Remaining Budget ({department_selection.upper()}) : ${amount_remaining:>10,.2f}")
    print("-"*30)
    return
```

These two pieces of information are finally passed through a print function to be displayed as output for the user.

```
-----
BUDGET RECORDS (OCT)
Beginning Budget (HUMAN RESOURCE): $  2,000.00
Remaining Budget (HUMAN RESOURCE): $  2,000.00
-----
```

Budget Editing

The budget every month for each department of finance, marketing, sales and human resources are set to be \$5,000, \$10,000, \$2,000 and \$2,000 respectively by default. This information is stored in a dictionary called *departmental_monthly_budget*, with the department names as keys and their corresponding beginning budget as their value.

```
departmental_monthly_budget = {"finance":5000, "marketing":10000, "sales":2000,
                               "human resource":2000}
```

Spend & Track allows managers and senior managers to make changes to the budget for each month. This is done via calling the *edit_budget()* function.

```
elif user_input == "3":
    edit_budget()
```

To ensure strong internal controls, employees who hold a position ranked below 'manager' will not be authorised to edit the budget. Such a simple verification process is done via storing the user's position in a variable called *pos*, and checking if *pos* is either "*manager*" or "*senior manager*".

Additionally, apart from those working in the finance department, Spend & Track restricts employees to only be able to edit the budget for their own department. As for those in the finance department, they have the authority to edit the budget for any of the four departments. This caters to the job scope of employees in the finance department, who may require such authority to carry out adjusting entries for other departments. Hence, the nested IF function is used given that there are two conditions to fulfil in this case.

```
def edit_budget():
    dept = employees_information[employee_ID_new]["Department"]
    pos = employees_information[employee_ID_new]["Position"]
    if pos == "manager" or pos == "senior manager":
        if dept == "finance":
            while True:
                edit_department = input("Which department would you like to edit the beginning budget for?\n")
                if edit_department.lower() in departments:
                    break
                else:
                    print("You have entered an invalid department. Please try again.")
            else:
                edit_department = dept
```

Should an intern or associate choose option "3" to edit the budget, Spend & Track will block this action by displaying an error message as they are unauthorised to make changes to the budget. The error message is shown below.

```
ERROR:
You are not authorized to perform this action. Only employees
ranked 'manager' and above are allowed to edit the budget.
```

Meanwhile, managers and senior managers would be able to proceed with editing the budget. They must first specify which month in particular they would like to edit the budget for. This month is assigned to a variable called *edit_month*. Next, the user must input the new budget amount. This piece of information will be assigned to a variable called *edit_amount*.

Upon confirmation by the user, the new budget amount will be updated in the *departmental_budget_balance* dictionary. This is done by calculating the net change in budget amount (which is assigned to a variable called *net_change*). The net change is then added to both the beginning and remaining budget balance so as to reflect the new budget changes.

```

if confirmation == "Y":
    net_change = edit_amount - departmental_budget_balance[edit_month][edit_department]["Beginning"]
    departmental_budget_balance[edit_month][edit_department]["Beginning"] += net_change
    departmental_budget_balance[edit_month][edit_department]["Remaining"] += net_change

```

The updated budget report will be displayed to the user as follows:

```

-----
SALES DEPARTMENT (UPDATED):
Beginning Budget: (JAN): $ 3,000.00
Remaining Budget: (JAN): $ 3,000.00

```

Keying in New Expense Entry

The key feature of the Spend & Track would arguably be the expense entry feature. This feature is defined in the `expense_entry()` function and can be called by any employee.

```

def expense_entry():
    # Obtaining department
    user_dept = employees_information[employee ID new]["Department"]

```

Employees under the finance department have the authority to enter expense entries for any department. This would cater to the job scope of finance employees, as they may be required to carry out expense recording functions for the entire company.

On the other hand, employees in the sales, marketing and human resource departments would only be able to record expenses for their own departments. Users have to enter a series of inputs in response to a series of prompts.

From these inputs, information regarding the department, expense category, expense date and expense amount are assigned to variables, namely `expense_department`, `expense_category`, `spending_date` and `expense_amount` respectively. Notably, the expense category must fall under one of the items in the `expense_categories` list.

```

expense_categories = ["Stationery", "Salary", "Marketing", "Miscellaneous", "Others"]

```

The programme will then check the expense amount against the budget balance. This is done by calling the `budget_balance_updater_temp()` function. This function returns two boolean values which are assigned to the variables `running_low` and `exceed` to indicate if the expense entry will cause the budget to run low or be exceeded respectively.

```

def budget_balance_updater_temp(dept,mth,amt):
    departmental_budget_balance_temp = departmental_budget_balance[mth][dept]["Remaining"] - amt
    running_low = departmental_budget_balance_temp <= 0.05*departmental_budget_balance[mth][dept]["Beginning"]
    exceed = departmental_budget_balance_temp < 0
    return (running_low, exceed)

```

Depending on the Boolean values assigned to these two variables, one of the following three scenarios may unfold:

1. Expense exceeds budget balance

An error message will be displayed, rejecting the expense entry. The user will then be brought back to the main menu.

2. Expense entry causes low budget balance

Our team has defined the budget balance to be running low when it is less than or equals to 5% of the beginning budget. Should the expense entry cause budget balance to be running low, then a warning message will be flashed to the user. The expense entry can still be recorded if desired.

3. Expense entry is well within budget balance

Should the expense entry not fall within the 5% threshold as indicated previously, then the expense entry can be considered to be well within the budget balance. In this case, the user can confirm the recording of the entry.

Upon confirmation of the expense entry, information relating to the expense entry will be stored into the *expense_history* dictionary. This is done by passing the expense details (department, category, spending date, expense amount, and entry date) into the *store_transaction()* function.

```
trans_code = store_transaction(expense_department,expense_category,spending_date,expense_amount,entry_date)
```

```
def store_transaction(dept,cat,trc_date,amt,ent_date):  
    details = [dept,cat,trc_date,amt,ent_date]  
    transaction_code = len(expense_history)  
    expense_history[transaction_code] = details  
    return transaction_code
```

The *store_transaction()* function takes in the five aforementioned pieces of information as arguments and returns a unique transaction code. This transaction code is based on the length of the *expense_history* dictionary, which is essentially the number of existing expense records. The unique transaction code will consequently be generated and displayed to the user. In addition, the budget balance is updated and displayed to the user.

```
Would you like to confirm the recording of this expense entry? Y/N  
y  
Transaction Code: 1
```

```
The budget balance for the sales department has been updated.  
There is $1,700.00 remaining for the month of MAR.
```

Editing Existing Expense Entry

In certain situations, employees in the finance department may need to rectify errors in expense entries. The `edit_existing_expense_entry()` function allows them to do so.

Spend & Track limits editing of the expense entries only to the finance department exclusively. Hence, if employees from the marketing, sales or human resource department select option "4" from the menu to edit existing expense entries, Spend & Track will block the action by displaying the message below.

```
You are not authorized to edit expense entries. Please contact
the finance department for further assistance.
```

Meanwhile, employees from the finance department would be able to proceed with editing the expense entry. The user will be asked to enter the transaction code of the particular entry that needs to be edited, which was generated and displayed to the user after they have successfully keyed in their expenses earlier. Spend & Track would be able to look up that transaction code which was stored in the keys of the `expense_history` dictionary.

```
while True:
    try:
        transaction_code = int(input("Please enter transaction code: "))
        if transaction_code in expense_history.keys():
            break
        else:
            print("This transaction code does not exist. Please try again.")
    except ValueError:
        print("Invalid transaction code entered. Please try again: ")
```

If the transaction code exists, the original expense details would be displayed to the user.

```
Please enter transaction code: 1
-----
EXISTING EXPENSE ENTRY:
    Transaction Code: 1
    Department: sales
    Expense Category: Salary
    Spending Date: 22 MAR
    Amount: $    500.00
    Entry Date: 07 October 21
-----
```

Thereafter, the user will be asked for confirmation to edit the entry. Upon successful confirmation, the new expense amount will be assigned to the variable *edit_amount*. The net change amount would be calculated and assigned to the variable *net_change_amount*. This value would then be passed through the *budget_balance_updater_temp()* function (as elaborated in the “Entering a new expense entry” section) to check if the updates would cause the budget to be exceeded. If it does, then Spend & Track will flag out these effects of the changes to the expense entry.

```
try:
    edit_amount = float(input("Enter new expense amount: $"))
    break
except ValueError:
    print("You have entered an invalid value. Please try again.")
net_change_amount = edit_amount - expense_history[transaction_code][3]
(running_low, exceed) = budget_balance_updater_temp(expense_history[transaction_code][0],
                                                    expense_history[transaction_code][2][-3:], net_change_amount)
if exceed:
    print("Adjustment has failed. Expense entry will exceed budget.")
    return
elif running_low:
    print("-"*30)
    print("WARNING: Budget is running Low.")
    print("-"*30)
```

If there are no issues, the new expense amount and new entry date would be updated in the *expense_history* dictionary. The remaining budget balance will also be reduced by the *net_change_amount*.

```
# Update Data
departmental_budget_balance[expense_history[transaction_code][2][-3:]]
[expense_history[transaction_code][0]]["Remaining"] -= net_change_amount

expense_history[transaction_code][3] = edit_amount
expense_history[transaction_code][4] = datetime.datetime.now().strftime("%d %B %y")
```

After the expense changes are made successfully, the updated expense entry would be displayed to the user.

```
Enter new expense amount: $1888
-----
UPDATED EXPENSE ENTRY:
    Transaction Code: 1
    Department: sales
    Expense Category: Salary
    Spending Date: 22 MAR
    Amount: $ 1,888.00
    Entry Date: 07 October 21
-----
```


Exporting Data (CSV)

Spend & Track would provide options for users to export the data that they need. The data to be exported includes the “Monthly expense history” where a csv file of all the expense transactions logged as well as its totals would be generated. An “Employee data” csv file of all login credentials of the employees can also be generated. In order to facilitate the exporting of data, our team has utilised the Pandas module.

However, access would be limited to certain groups of people for the data to be exported. This control ensures that sensitive information would not be obtained by irrelevant personnels. In the case where the user provided a response that does not export either of those data, Spend & Track will prompt the user to key in again until the correct response is received. This is done through a while-loop checker.

```
while export_selection not in ["1","2"]:  
    export_selection = input("Invalid response received. Please try again: ")
```

```
What data would you like to export?  
[1] Monthly expense history  
[2] Employee data  
  
6  
  
Invalid response received. Please try again: 7  
Invalid response received. Please try again: 6  
Invalid response received. Please try again: 5  
Invalid response received. Please try again: 4  
Invalid response received. Please try again: 1  
  
Which department's data would you like to export?
```

Exporting “Monthly expense history”:

If a non-finance employee chooses to export the “Monthly export history”, they would only be able to export the expenses history for their own department. On the other hand, a finance employee would have the option to export the “Monthly export history” of other departments into csv files.

If a *non-finance* employee exports the monthly export history:

```
[2] Key in an expense entry
[3] Edit budget
[4] Edit existing expense entry
[5] Export data
[6] End programme
5

What data would you like to export?
[1] Monthly expense history
[2] Employee data
1

Which month would you like to export data for? Please enter your
selection in MMM format.
jan
Your data has been exported.
```

If a *finance* employee were to export the monthly export history:

```
[3] Edit budget
[4] Edit existing expense entry
[5] Export data
[6] End programme
5

What data would you like to export?
[1] Monthly expense history
[2] Employee data
1

Which department's data would you like to export?
[1] Finance
[2] Marketing
[3] Sales
[4] Human Resource|
```

A while-loop checker is also placed here in the case where the user chooses a departmental option that is not stated in the menu.

```
while expense_department_export not in ["1","2","3","4"]:
    expense_department_export = input("Invalid response received. Please try again: ")
```

```
Which department's data would you like to export?
[1] Finance
[2] Marketing
[3] Sales
[4] Human Resource
7

Invalid response received. Please try again: 8

Invalid response received. Please try again: 4

Which month would you like to export data for? Please enter your
selection in MMM format.
```

Spend & Track would also prompt the employee to re-enter the month if the employee enters an input not in the MMM format for months. This is done through a while-loop checker.

```
expense_month_export = input("Which month would you like to export data for? Please enter your selection in MMM f
while expense_month_export not in months:
    expense_month_export = input("You have entered an invalid month. Please enter the month in which the expense i
```

In order to export the monthly expense history, our team first constructed a dataframe from the dictionary *expense_history*, which contains the information of the various expense entries that has been previously keyed in before exporting the dataframe to a csv file. This dataframe is named *df_expense_report*.

The programme assigns the department for the expense entries to be exported to a variable called *expense_department_export*. The month which expense entries are to be exported from are also assigned to a separate variable called *expense_month_report*.

Thereafter, the dataframe will be modified by first looking through the 'department' column and identifying all rows which match the selected department. Next, it will look through the 'spending_date' column using the *str_contains* method to identify all rows containing the selected month.

Finally, a row is added at the bottom of the dataframe to display the sum of all expense amounts in the dataframe.

```
df_expense_history = pd.DataFrame.from_dict(expense_history, orient='index', columns=["department", "category", "spending_date", "amount", "entry_date"])
df_expense_export = df_expense_history[df_expense_history["department"]==expense_department_export]
df_expense_export = df_expense_export[df_expense_export["spending_date"].str.contains(expense_month_export)]
df_expense_export.loc['Total'] = pd.Series(df_expense_export['amount'].sum(), index = ['amount'])

df_expense_export.to_csv(f"{expense_department_export}_{expense_month_export}_expenses.csv")
print("Your data has been exported.")
```

If the data has been successfully exported into a csv file, a message showing that the data has been exported will be generated and users can then open up the csv file to view the expense entries. The first column in the csv file represents the transaction code of the expense entries.

	department	category	spending_date	amount	entry_date
1	sales	Stationery	12-Jan	1000	7-Oct-21
2	sales	Marketing	21-Jan	130.5	7-Oct-21
Total				1130.5	

Exporting “Employee data”:

This data file can only be exported by HR employees. If a non-HR employee were to export employee data, they would not be able to continue and the action would be blocked.

You are not allowed to export employee data. Please contact the human resource department.

Similar to the exporting of monthly expense history, a dataframe is first created based on the dictionary containing the information of the employees (see screenshot on page 6) in the organisation before exporting the dataframe to a csv file. The dataframe called *df_employees_info* is then subsequently transposed so that each row will reflect the various information based on the employee ID and each column represents the password, name, department and position.

```
else:
    if user_dept == "human resource":
        df_employees_info = pd.DataFrame(employees_information)
        df_employees_info = df_employees_info.T
        df_employees_info.to_csv("employees_info.csv")
        print("Your data has been exported.")
```

Before transposing:

	s01	f02	h03	m04
Password	password1	password2	password3	password4
Name	Sandra	Fred	Hazel	Mabel
Department	sales	finance	human resource	marketing
Position	manager	senior manager	associate	intern

After transposing:

	Password	Name	Department	Position
s01	password1	Sandra	sales	manager
f02	password2	Fred	finance	senior manager
h03	password3	Hazel	human resource	associate
m04	password4	Mabel	marketing	intern

After transposing the dictionary, the employee IDs are easily identifiable as the indexes of the dataframe.

Again, if the data has been successfully exported into a csv file, a message showing that the data has been exported will be generated and users can then open up the csv file to view the employee information. The first column of the exported csv file represents the employee ID of the various employees.

	Password	Name	Department	Position
s01	password1	Sandra	sales	manager
f02	password2	Fred	finance	senior manager
h03	password3	Hazel	human resource	associate
m04	password4	Mabel	marketing	intern

User Manual

[1] Logging In:

For new users:

1. To begin, indicate that you are a new user if you do not have an existing account. Input "Y"
2. Enter your employee ID. Input "s02".
3. Enter your preferred password. Please remember to set a strong password of at least 8 characters for security purposes. Input "password5".
4. Re-enter your password for confirmation. Input "password5" again. A message showing that your password has been created should appear.
5. Enter your name, position and department. Input the following:

Employee Name	Jane
Employee Position	Manager
Employee Department	Sales

6. After keying in your name, position and department, an employee record summary containing your account details will be displayed as a confirmation that your account has been created.

```
Welcome! Are you a new user? Y/N Y
Please enter your new employee ID: s02
Please enter your new password: password5
Please confirm your new password: password5
Password created successfully!

What is your name? Jane

Please enter your position (Intern/Associate/Manager/Senior Manager) Manager

Please enter your department (Finance/Marketing/Sales/Human Resource) Sales
=====
Employee Record Summary:
=====
Employee ID: s02
Password: password5
Employee Name: Jane
Employee Position: manager
Employee Department: sales
```

7. Upon creation of an account, the main menu of Spend & Track will be displayed.

```
Welcome, Jane!

What would you like to do today?
[1] View budget
[2] Key in an expense entry
[3] Edit budget
[4] Edit existing expense entry
[5] Export data
[6] End programme
Selection: |
```

For existing users:

1. If you have an existing account on Spend & Track, indicate that you are not an existing user to begin the login process. Input "N".
2. Enter your employee ID. Input "f02".
3. Enter your password. Input "password2".
4. Upon successful login attempt, the main menu of Spend & Track will be displayed.

```
Welcome! Are you a new user? Y/N  N

Please enter your employee ID: f02

Please enter your password: password2
Password is correct
-----
Welcome, Fred!

What would you like to do today?
[1] View budget
[2] Key in an expense entry
[3] Edit budget
[4] Edit existing expense entry
[5] Export data
[6] End programme
Selection: |
```

[2] View Budget:

1. To view the budget, select option 1 from the main menu. Input "1".
2. You will then be prompted to enter the month you would like to view the budget for. Please do so in **MMM** format. Input "oct".

```
Please enter the month you would like to view in MMM format: oct
```

For employees NOT IN the finance department:

You only have the authority to view the budget for your own department. The budget records for your department will be automatically retrieved and displayed. You can view the beginning budget as well as the remaining budget.

```
-----  
BUDGET RECORDS (OCT)  
Beginning Budget (HUMAN RESOURCE): $ 2,000.00  
Remaining Budget (HUMAN RESOURCE): $ 2,000.00  
-----
```

You will then return to the main menu.

For employees IN the finance department:

You have the authority to view the budget for all departments. The budget records for all departments will be automatically retrieved and displayed. You can view the beginning budget as well as the remaining budget for all departments.

```
-----  
BUDGET RECORDS (OCT)  
Beginning Budget (FINANCE): $ 5,000.00  
Remaining Budget (FINANCE): $ 5,000.00  
Beginning Budget (MARKETING): $ 10,000.00  
Remaining Budget (MARKETING): $ 10,000.00  
Beginning Budget (SALES): $ 2,000.00  
Remaining Budget (SALES): $ 2,000.00  
Beginning Budget (HUMAN RESOURCE): $ 2,000.00  
Remaining Budget (HUMAN RESOURCE): $ 2,000.00  
-----
```

You will then return to the main menu.

[3] Edit Budget:

For interns or associates:

You are not authorised to make any changes to the budget. Hence, an error message will be displayed.

```
ERROR:
You are not authorized to perform this action. Only employees
ranked 'manager' and above are allowed to edit the budget.
```

You will then return to the main menu.

For managers and above (non-finance department):

You are authorised to make changes to the budget of your own department.

1. Enter the month you would like to edit the budget for (Note: Month in **MMM** format). The current beginning budget will be displayed. [Input "jan"](#).
2. Enter the desired new budget amount. [Input "3000"](#).
3. Your desired changes will be displayed. Indicate if you would like to confirm the changes. [Input "y"](#).
4. Upon confirmation, the updated beginning budget and remaining budget for that month will be displayed. You will then return to the main menu.

```
Which month would you like to edit the budget for? Please enter your selection in
MMM format: jan
-----
SALES DEPARTMENT
Beginning Budget (JAN): $ 2,000.00
-----

What would you like to change the amount to?
New Beginning Budget: $3000

This will be the new budget for the sales department:

Beginning budget (JAN): $ 3,000.00

Do you wish to confirm the changes? Y/N
y
-----
SALES DEPARTMENT (UPDATED):
Beginning Budget: (JAN): $ 3,000.00
Remaining Budget: (JAN): $ 3,000.00
-----
```


For managers and above (finance department):

You are authorised to make changes to the budget for any of the departments.

1. Enter the department you would like to edit the budget for. Input "marketing".
2. Enter the month you would like to edit the budget for (Note: Month in **MMM** format). The current beginning budget will be displayed. Input "jan".
3. Enter the desired new budget amount. Input "11000".
4. Your desired changes will be displayed. Indicate if you would like to confirm the changes. Input "y".
5. Upon confirmation, the updated beginning budget and remaining budget for that month will be displayed. You will then return to the main menu.

```
Which department would you like to edit the beginning budget for?
marketing

Which month would you like to edit the budget for? Please enter your selection in
MMM format: jan
-----
MARKETING DEPARTMENT
Beginning Budget (JAN): $ 10,000.00
-----

What would you like to change the amount to?
New Beginning Budget: $11000

This will be the new budget for the marketing department:

Beginning budget (JAN): $ 11,000.00

Do you wish to confirm the changes? Y/N
y
-----
MARKETING DEPARTMENT (UPDATED):
Beginning Budget: (JAN): $ 11,000.00
Remaining Budget: (JAN): $ 11,000.00
-----
```

[4] Entering a new expense entry:

1. To enter a new expense entry, select option 2 from the main menu. Input "2".
2. You will then be prompted to enter the month you would like to view the budget for. Please do so in **MMM** format. Input "oct".

For employees NOT IN the finance department:

You are only authorized to record expenses for your own department. Proceed to Step 2 of this section as elaborated below.

For employees IN the finance department:

You are authorized to record expenses for all departments.

1. Specify the department which you are recording the expense entry for (either sales / marketing / human resource / finance). Input "sales".

```
Which department would you like to enter the expense item for?  
sales
```

2. Select the category of expense. Input "5".

```
Please select the category of expense:  
1) Stationery  
2) Salary  
3) Marketing  
4) Miscellaneous  
5) Others  
Selection: 5
```

3. Key in the month in which the expense was incurred. Do so in **MMM** format. You will then be prompted to key in the date on which the expense was incurred. Do so in **DD** format. The transaction date will be displayed to you. Input "sep" and "30".

```
Please enter the month in which the expense was incurred in MMM format: sep  
Please enter the date on which the expense was incurred in DD format: 30  
Transaction Date:      30 SEP.
```

4. Key in the amount of expense you would like to record.

5a. If the amount entered exceeds the monthly departmental budget, you will encounter an error message.

```
What is the amount of expense you would like to record? 3000
Entry has failed. Expense entry will exceed budget.
```

5b. If the expense entry would cause the budget to run low (defined to be less than 5% of the beginning budget), a warning message will be displayed. A summary of the expense entry will also be displayed. Indicate if you would like to continue with the recording of the entry.

```
What is the amount of expense you would like to record?
Amount of Expense: $ 1900
-----
WARNING: Budget is running low.
-----

SUMMARY OF EXPENSE ENTRY:

Department: SALES
Category: OTHERS
Spending Date: 30 SEP
Amount: $ 1,900.00

Recorded by: Fred
Date: 07 October 21

-----

Would you like to confirm the recording of this expense entry? Y/N
```

5c. If the expense entry does not cause the budget to run low, a summary of the expense entry will be displayed. Indicate if you would like to confirm the entry.

What is the amount of expense you would like to record?

Amount of Expense: \$ 500

SUMMARY OF EXPENSE ENTRY:

Department: SALES

Category: OTHERS

Spending Date: 30 SEP

Amount: \$ 500.00

Recorded by: Fred

Date: 07 October 21

Would you like to confirm the recording of this expense entry? Y/N

6. Upon confirmation of the entry, a transaction code will be generated and displayed. The remaining departmental budget for the month will also be calculated and displayed. You will then return to the main menu.

Would you like to confirm the recording of this expense entry? Y/N

y

Transaction Code: 2

The budget balance for the sales department has been updated.

There is \$1,500.00 remaining for the month of SEP.

[5] Editing an existing expense entry:

For employees NOT IN the finance department:

You are not authorised to make changes to the existing expense entries. Hence, an error message will be displayed.

```
You are not authorized to edit expense entries. Please contact
the finance department for further assistance.
```

For employees IN the finance department:

You are authorised to make changes to the existing expense entries.

1. Enter the transaction code of the expense entry that needs to be edited. Input "2".
2. Check the entry displayed and input "Y" to continue. Input "y".

```
Please enter transaction code: 2
-----
EXISTING EXPENSE ENTRY:
  Transaction Code: 2
  Department: sales
  Expense Category: Others
  Spending Date: 30 SEP
  Amount: $    500.00
  Entry Date: 07 October 21
-----
Do you wish to edit the entry? Y/N
|
```

3. Enter the new expense amount.
- 4a. If the amount entered exceeds the monthly departmental budget, you will encounter an error message.

```
Enter new expense amount: $70000
Adjustment has failed. Expense entry will exceed budget.
-----
```

- 4b. If the adjustment would cause the budget to run low (defined to be less than 5% of the beginning budget), a warning message will be displayed. A summary of the updated expense entry will also be displayed.

```
Enter new expense amount: $1950
-----
WARNING: Budget is running low.
-----
UPDATED EXPENSE ENTRY:
    Transaction Code: 2
    Department: sales
    Expense Category: Others
    Spending Date: 30 SEP
    Amount: $ 1,950.00
    Entry Date: 07 October 21
-----
```

4c. If the adjustment does not cause the budget to run low, a summary of the updated expense entry will be displayed.

```
Enter new expense amount: $600
-----
UPDATED EXPENSE ENTRY:
    Transaction Code: 2
    Department: sales
    Expense Category: Others
    Spending Date: 30 SEP
    Amount: $ 600.00
    Entry Date: 07 October 21
-----
```

5. After the adjustments have been made, you will return to the main menu.

[6] Exporting data into a CSV file:

1. To export data, select option 5 from the main menu. [Input "5"](#).

```
What would you like to do today?
[1] View budget
[2] Key in an expense entry
[3] Edit budget
[4] Edit existing expense entry
[5] Export data
[6] End programme
5
```

2. Next, an option will be provided to export either the **"Monthly expense history"** or **"Employee data"** into a csv file.

- Select option 1 to export Monthly expense history [Input "1"](#)
- Select option 2 to export Employee data [Input "2"](#).

```
What data would you like to export?
[1] Monthly expense history
[2] Employee data
1
```

Exporting **"Monthly expense history"**:

For employees IN the finance department:

You will be given the choice to export the expense history for any of the departments. [Input "2"](#).

```
Which department's data would you like to export?
[1] Finance
[2] Marketing
[3] Sales
[4] Human Resource
2
```

Next, you will be prompted to key in the month of the expense data to be exported in MMM format. [Input "jan"](#).

```
Which month would you like to export data for? Please enter your
selection in MMM format.
jan
```

The data will then be exported.

For employees NOT IN the finance department:

You will only be able to export the expense history for your own department after keying in the month to be exported in MMM format. [Input "1" and "jan"](#).

```
What data would you like to export?
[1] Monthly expense history
[2] Employee data

1

Which month would you like to export data for? Please enter your
selection in MMM format.
jan
Your data has been exported.
```

Exporting “employee data”:

For employees NOT IN the HR department:

Inputting option 2 will not allow you to export the employee data and a restriction will be shown.

```
What data would you like to export?
[1] Monthly expense history
[2] Employee data

2
You are not allowed to export employee data. Please contact the human
resource department.
```

For employees IN the HR department:

Selecting option 2 will start the export of the employee data. [Input "2"](#).

```
What data would you like to export?
[1] Monthly expense history
[2] Employee data

2
Your data has been exported.
```


[7] Exit Programme:

1. To exit, select option 6 from the main menu. [Input "6"](#).

```
Welcome, Fred!  
  
What would you like to do today?  
[1] View budget  
[2] Key in an expense entry  
[3] Edit budget  
[4] Edit existing expense entry  
[5] Export data  
[6] End programme  
Selection: 6  
Thank you and have a nice day!
```