University of
Science and Technology of Hanoi

Advanced Programming with Python
Final Project Report

Electricity Information Management System

Group 6 - BI10 | ICT & MATH

May 2021

Contents

1 Introduction

1.1 Overview

Since we are not living in the stone age anymore, electricity is very essential in our lives. We do this project with electricity, you are reading this report with electricity. The electricity we are using was supplied by electricity companies (for example, in Vietnam, we have EVN) and they require high-quality electricity information management systems for best performance and efficiency. In this project, we try to simulate this real-life situation and create a system called EIMS that can satisfy those requirements.

1.2 Group members

Dương Đăng Hưng BI10-073

Tạ Quang Hiếu BI10-065

Thẩm Như Phong BI10-136

Nguyễn Hoàng Yến BI10-197

1.3 What is our program and what it does?

EIMS is programmed using Python, a high-level programming language with convenient modules and packages. With this system, the administrator can:

- Insert, update and delete information of households and areas.
- He/she can also input the measurement of a specific household's electricity meter, and the system will automatically calculate the consumption for that month and the cost to be paid.
- The system also supports updating the payment status to classify which household has paid, which household has not, for how long a specific household has not paid the bills, etc.

To sum up, this system gives you every basic functionality for an electricity management system.

1.4 Why should you use our program?

Our design is based on realistic requirements of the electric power management system that all innovative electricity companies expect. After inquiring today's existing electricity management systems, we created our program with some significant advantages:

- Firstly, our program works reliably. It can properly solve some fundamental managing functions, such as 'insert', 'update', 'display', and 'delete' data. Moreover, the system can automatically calculate electrical consumption and electrical costs while reading users' electricity meters monthly. Furthermore, the system offers basic statistic help such as listing users, monthly consumption and suggest a user's specific history record.

- Secondly, we approached the system from the perspective of our main customers (electric power supply companies) i.e., likely less familiar with information and communication technology. Therefore, we designed our system with a Graphical User Interface (GUI) using Tkinter package. It is a graphical interface visualizing the communication with the user to allow for easy interaction with the system. The common User Interface includes Graphical representations like buttons and icons rather than using text-based or command-based interactions. Consequently, the usability of our system is much more friendly to its customers.

- Thirdly, we built a logical database structure that supports collecting organized information. As mentioned earlier, managers can access, manage and update information with a user-friendly interface. The database system contributes to the business development of an organization by communicating information related to its users, areas, meter records, and history records. With statistic records, an electric power supply organization will be able to predict user's consumption and its ability to support future requirements. As a result, the organization will be able to increase and maximize its profit.

- Finally, our system is realistic. Initially we examined selected organizations that are taking part in the business of electricity supply, especially EVN (Vietnam Electricity). We then investigated how households, factories or other users calculate their consumption, cost and what they expect from suppliers. Later, we outlined what attributes and methods the management system should have and why the system

needed them. Furthermore, we focused on how the system addressed all the requirements and tested it to prove a working program for the customer. To sum it up, we made an applicable system.

2 How we created it ?

2.1 Entity-relationship Diagram and Database Schema

Firstly, before starting to code anything, we have to design how our program will look like, which entity sets will be created, and what attributes they possesses. Below is the entity-relationship diagram (ERD) of our program:

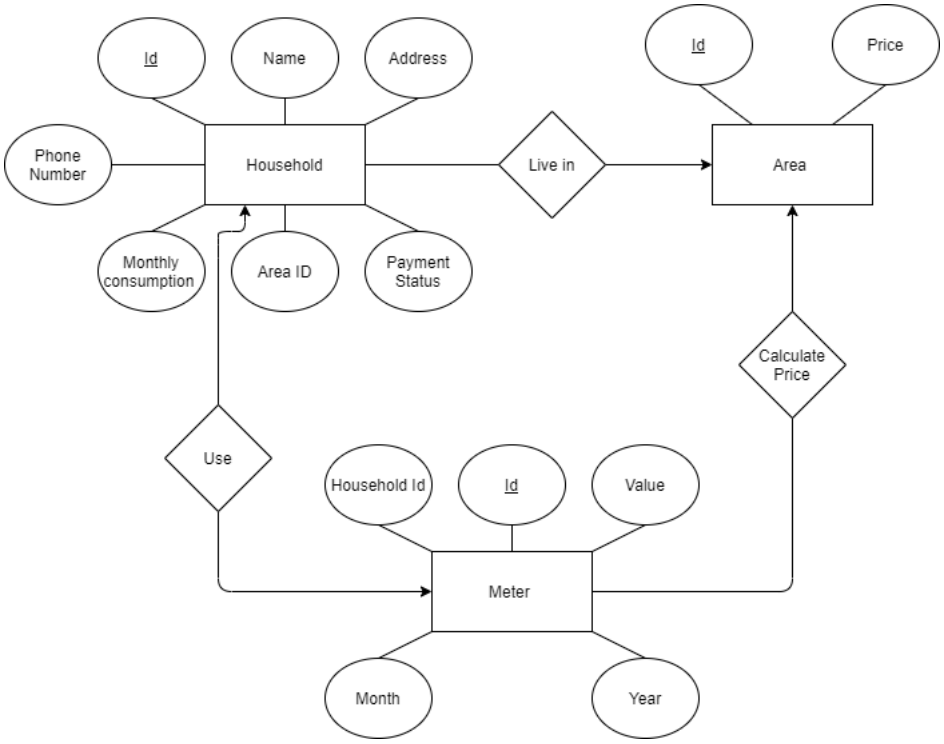

Figure 1: The ERD

As you can see here we have 3 entity sets in total: *Household*, *Area* and *Meter*. To discuss them in more detail:

- **Household**: An entity from this entity set has 7 attributes in total with the primary key *Id* and a foreign key *Area ID* referenced to the

Id of the Area. This entity set represents the users who consume the electricity provided by our company.

- **Area**: An entity from this entity set has 2 attributes in total with the primary key *Id*. This entity set represents the areas where users currently live.

- **Meter**: An entity from this entity set has 5 attributes in total with the primary key *Id* and a foreign key *Household ID* referenced to the *Id* of the Household. This entity set helps us to calculate how much the users have consumed and how much they have to pay.

Also in the ERD, we have 3 relationships:

- **Live in**: The relationship between **Household** and **Area**. Since an area can contains many households, and one household can only be in 1 area, it is a many-to one-relationship .

- **Use**: The relationship between **Household** and **Meter**. Since one household can only have one meter, and one meter belongs to one household only, it is a one-to-one relationship.

- **Calculate price**: The relationship between **Meter** and **Area**. Since an area can contains many meters, and one meter belongs to one area only, it is a many-to-one relationship.

To have more understanding about the relationship we have developed this schema:

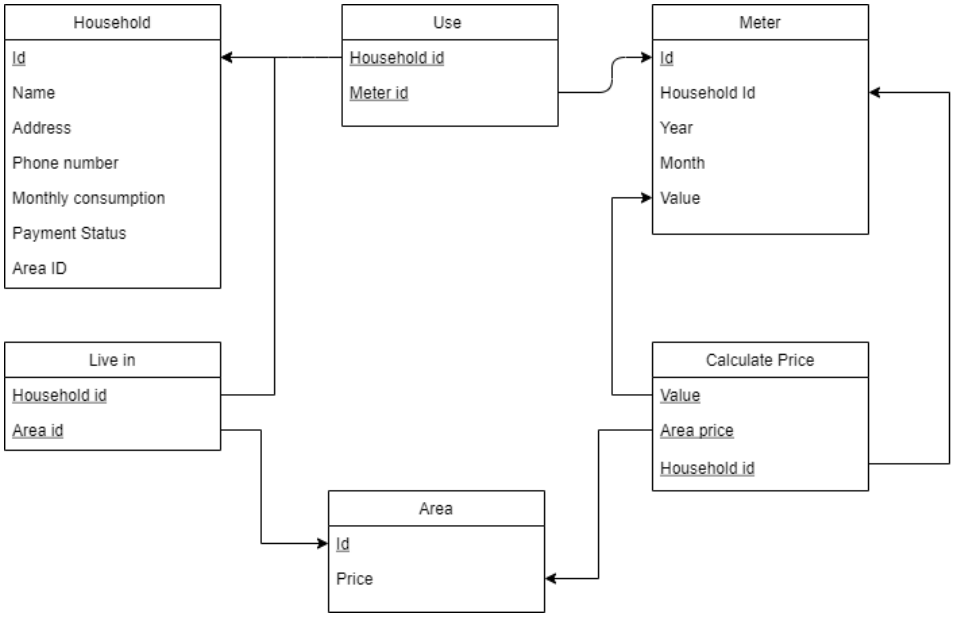

Figure 2: The Database Schema

From this schema we can easily see the detailed relationships between the entity sets and develop the functions in our program:

- **Use**: This relationship is based on *Household_id* and *Meter_id*. It will be used to look up for *Household_id*.
- **Live in**: This relationship is based on *Household_id* and *Area_id*. It will be used to look up for *Area price* through *Area_id*.
- **Calculate price**: This relationship is based on *Household_id*, *Value* and *Area price*. The *Household_id* here is a foreign key derived from **Household**. After we have found the *Area price* using *Household_id* through *Live in* relationship, we can directly multiply it to the *Value* to calculate the bill.

2.2 Program Structure

2.2.1 Modules

In our program, package "tkinter" is mainly used for GUI programming as it is the requirement of the project. Some basic modules are also used, such as:

- sqlite3: For database implementation. It's light, simple to use, SQL-based, perfectly suitable for medium-scale projects like this!
- tkinter.ttk: For more advanced decorating. In our program, we used TreeView from ttk to display information of the three main classes. TreeView also provides a great mean for the user to interact with the tuples dynamically.
- tkinter.messagebox: Mostly used to make announcements, such as show errors or show messages.
- strftime: To convert time to a string then display it at the top right corner of the windows. Also used for decorating.

2.2.2 Classes

From Section 2.1, we have seen that there are 3 main entity sets:

- Households
- Areas
- Meters

In our Python codes, we also have 3 main classes were created based on these main entity sets (with other supporting classes). But they perform a little bit differently. Particularly, each class represents a window/page that can be displayed on the user's screen, not only the entity sets. The widgets of the window are considered as the attributes of the class, while those events in the GUI (button pressing, TreeView scrolling, TreeView searching, etc.) are bound with the functions defined in the class:

- login_page: The login screen that will pop-up when the user start the program.
- Admin_Page:
  - The admin screen that the user could access if he/she login successfully.
  - Here, the user has 3 choices: "Households", "Areas" and "Meters". ("About us" would be developed in the future).
  - From this page onwards, the user can logout to get back to the login screen at any time.

- Households, Areas, Meters:
  - Represent the windows that will be displayed corresponding to 3 main choices.
  - Here, he/she can view the data via the TreeView, then continue to choose to search for a specific tuple in the TreeView, add/delete /update information in the database, exit to get back to the admin page or logout to shut the system down.
- add_households, add_areas, add_meters:
  - Represent the windows that will pop-up when the user chooses to add new data into households/areas/meters.
  - Here after he/she has filled all the required fields and press the button "Add", all the input datas will be stored into the database and the information in the TreeView will also be updated at the same time.
  - The user can also clear all the fields to re-input from the beginning.
- Update_Households, Update_Area:
  - Represent the windows which will pop-up when the user chooses one tuple in households/areas to update the information of that tuple.
  - Here, after he/she has filled all the required fields with updated information and press the button "Update", all the input datas will be updated to the database and the information in the TreeView will also be updated at the same time.
  - The user can also clear all the fields to re-input from the beginning.

2.2.3 Input and Output

There are plenty of occasions when the user runs the system, the system then takes some inputs via entries placed in the GUI using Tkinter, then return some outputs. We are now going to list all those cases:

- When the user starts the program and brought to the login screen:
  - First, the user will *input* username and password into corresponding entries and click "Login".

  - Afterwards, the system will search for the input username and password in table “Accounts” in the database.
  - If the system find that exact account in table “Accounts” with authority “Admin”, the system will inform (*output*) the user with a message: “The login is successful.” and move the user to the Admin Page.
  - Otherwise, the user will be informed with an error: "Incorrect username or password".

- When the user wants to search for a specific household/area/meter:
  - First, the user will *input* the ID of the household/area/meter required searching into the search query and click "Search".
  - Afterwards, the system will search for the tuple with that ID in the TreeView.
  - If the system can find that exact tuple in the TreeView, the system will then inform (*output*) the user with a message to confirm the successful searching and auto-select that tuple for the user.
  - Otherwise, the user will be informed with an error that the input ID was invalid.

- When the user wants to add new household/area/meter:
  - First, the user will *input* the required fields of the new household/area /meter and click “Add”.
  - Afterwards, the system will check if all input data are valid (e.g: blanks, incorrect data types, values out of range, non-exist foreign keys, etc.).
  - If all input data are valid, the system will then inform (*output*) the user with a message: “Household/Area/Meter successfully added into database” and new data will be displayed on the TreeView.
  - Otherwise, the user will be informed with a specific error for each case.

- When the user wants to update an existed household/area:
  - First, the user will choose a specific tuple to update information.
  - Afterwards, the system will check if the user is selecting only 1 tuple. If the user is selecting 2 or more, the system will inform with an error: “Can only update one household/area at a time”.

- If the update window pop-up, the user will update by *re-input* all the fields that required updating and click "Update".
- The system then check if all data are valid (the checking process is the same as when new data is added).
- If all updating data are valid, the system will then inform (*output*) the user with a message: "Household/Area successfully updated to database" and updated data will be displayed on the TreeView.
- Otherwise, the user will be informed with a specific error for each case.

2.3 User Interface Structure

As illustrated in section 2.2.1, everything related to the GUI was provided by tkinter, and now we will take a closer look at it.

Our system has a total of 10 windows, which we would like to call "pages":

- page1: Login page
- page2: Admin page
- page3: Households management page
- page4: Add household page
- page5: Areas management page
- page6: Add area page
- page7: Meters management page
- page8: Update area page
- page9: Update household page
- page10: Add meter page

To understand more about the relationships among these pages, please take a look at the figure below:
From this figure, we can see clearly about how the windows are arranged in order and how we can move through them. Each two-way arrow demonstrates how we can move back and forth between the windows connected with it. Let us divide the windows in our system into 4 different stages:

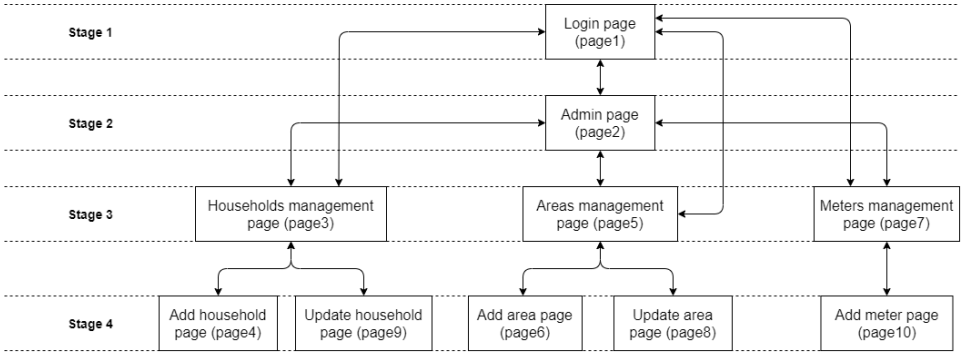

Figure 3: UI Windows Structure

- Whenever we move from a window in a higher stage to a lower one (considering their positions in the figure), the lower window - a Toplevel window, will pop-up and be displayed on the screen, while the higher-staged window will be "withdrawn" (except for stage 4, where the 3rd-staged window is not withdrawn but the 4th-staged window still pop-up).

- In the contrary, if we move back to the higher-staged window from a lower one (using Exit button, Logout button, or WM_DELETE_WINDOW), the lower-staged window will be completely destroyed, and the higher-staged window then "deiconify" and brought back to the screen.

Now for a more detailed view inside the windows. For aesthetic purpose, every window's background is already drawn using Photoshop, we then only need to place the widgets of that window at the right coordinates so that it perfectly fit the background. These are some images of windows from 4 different stages:

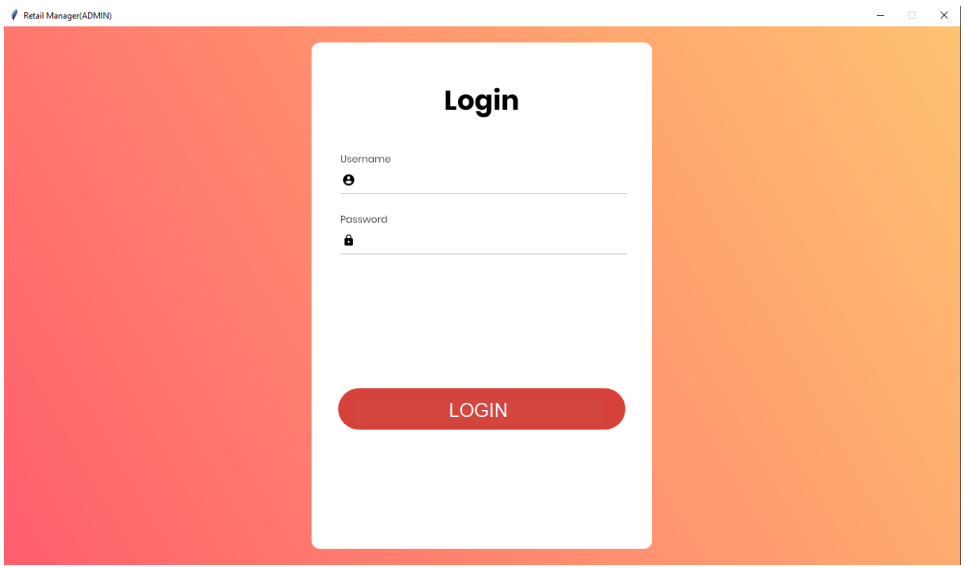

Figure 4: Login window (stage 1) with 3 widgets: a login button, a username entry and a password entry

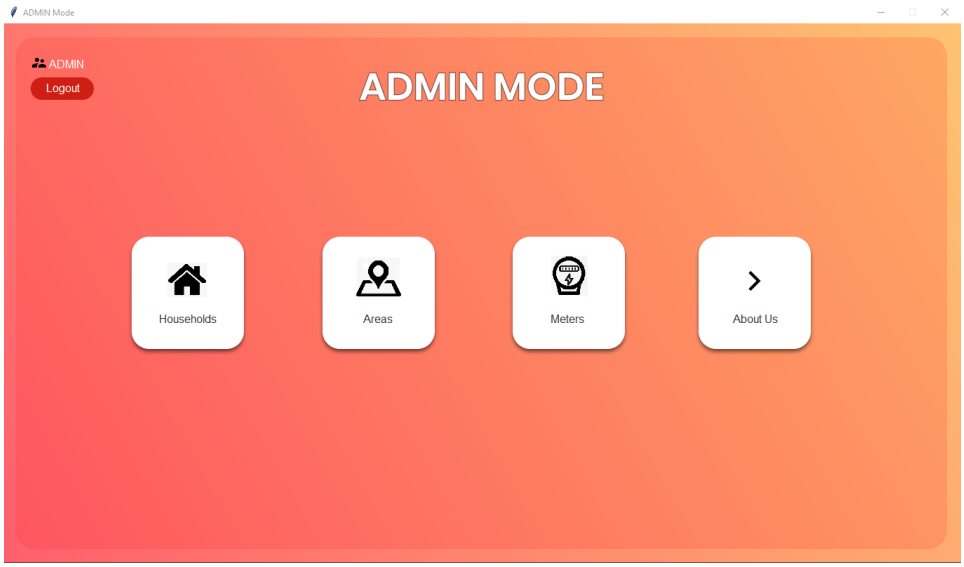

Figure 5: Admin window (stage 2) with 6 widgets: a label display “ADMIN”, a logout and 4 other buttons

Figure 6: Household window (stage 3) with 10 widgets: 2 labels display "ADMIN" and the clock, a TreeView, a search entry and 6 buttons: logout, search, add, update, delete, and exit

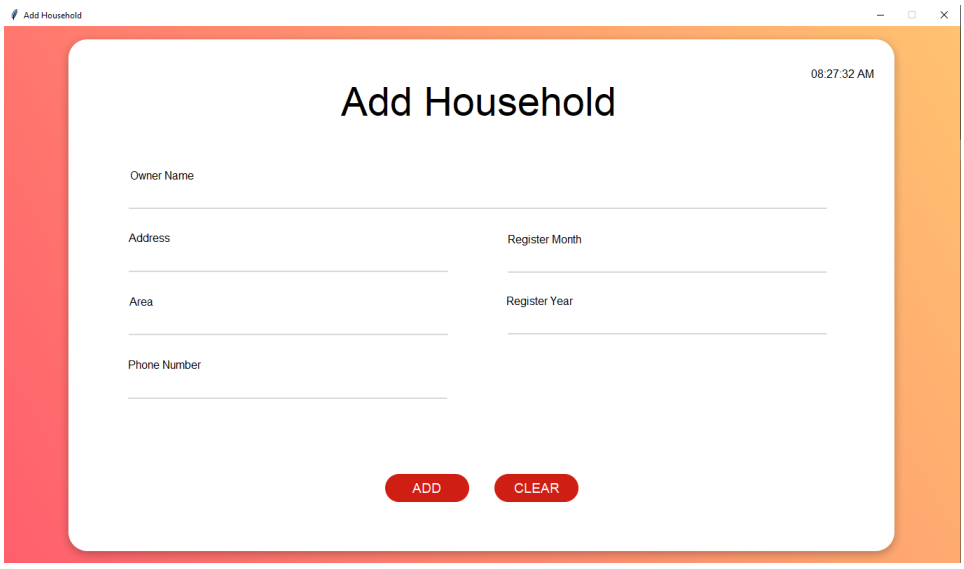

Figure 7: Add Household window (stage 4) with 9 widgets: 1 label display the clock, 6 entries corresponding to 6 fields, add and clear buttons

3 Conclusion

Above are all about our program and our progress to finish the system. To sum up, we have completed EIMS with:

- Providing Graphical User Interface (GUI) with Tkinter
- Database implementation with sqlite3
- Basic functions: add, update, delete, search and display.

References

[1] realmacaw: Retail Management
`https://github.com/realmacaw/real-mart`