

# ATLAS IOT APPLICATION IDE DEVELOPMENT & ASSESSMENT

## —— Report of Group 3

Jingfan, Wang 5191-0196

Zixun, Wang 3725-9823

Yixin, Wei 5114-6181

Lihuang, Xie 9089-4648

Junyi, Xie 9799-2838

## 1. INTRODUCTION

### 1.1 Brief Introduction of the Atlas IoT IDE

In this group project, we put in enough energy and time to develop an Atlas IoT application IDE(hereinafter referred to as Atlas IDE or IDE). This IDE is based on the Atlas IoT framework, Python Tkinter (an open source Python GUI framework) and VSS, with simple and clear GUI, and is able to help users develop and test simple Atlas IoT applications.

The IDE should be connected to the VSS and is able to capture tweets and learn information from VSS. It focuses on the user's own things. That is, it filters out useful tweets according to the user's settings.

The IDE has several functional pages including Initialization, Home Page, Things, Services, Relationships, Recipe and Apps.

- Initialization: designed for users to set up the whole IDE.
- Home Page: the main page of IDE and can switch to different pages.
- Things: to display basic information (including images) of specific things it learns during initialization.
- Services: to display all the available services of a user's things and can view services of a thing by setting the filter.
- Relationships: to display all the relationships of user's things and can view relationships of a thing by setting the filter.
- Recipe: easy and available for users to build a new Atlas application based on existing services and relationships through clicking operations.
- Apps: to display existing apps information and manage applications.

The IDE is implemented by Python 3 and is tested in Windows 10 OS with openvpn.

**Note:** unformatted tweets cannot be received because our IDE can only resolve formatted tweets.

### 1.2 Work Division

	Workload(1-5)	Jingfan Wang	Zixun Wang	Yixin Wei	Lihuang Xie	Junyi Xie
Initpage	5	40%	20%	40%		
Things	2		90%	10%		
Services	3					100%
Relationships	3				100%	
Recipe	4		100%			
Apps	2				10%	90%
App Manager	5	5%	5%	5%	75%	10%
Code Integration	4	60%		40%		
Testing	4	30%	10%	50%	5%	5%
Report Writing	3	30%	25%	15%	10%	10%

In Section 2, the implementation of the IDE will be introduced in detail.

In Section 3, we are going to display the results of our IDE and show some screenshots of the running process.

In Section 4, there will be a usability study of our IDE which is done by about 10 classmates outside our group.

In Section 5, we make a conclusion and put forward some improvement ideas of our work.

## 2. IMPLEMENTATION

In this section, we will give details of how we designed this Atlas IoT application IDE and what functions and features we have implemented.

### 2.1 Initialization

The Initialization page is designed for users to enter the necessary information (the smart space ID, IDs of things and IP addresses of each thing) to set up the Atlas App IDE. Then, the IDE will filter and capture corresponding tweets from VSS, extract useful data and save them into local files for subsequent use.

Before starting IDE, it is crucial for the user to make sure their things online and in the VSS, and input correct information into text boxes. The program will check automatically whether the number of the thing ids and the number of the IP addresses are equal. And check if the IP addresses satisfy the format of 10.254.0.X. If not, there will pop up a message box after the user clicks the start button to warn the user, and he/she should check the information he/she entered. After clicking the “Start” button, the IDE will freeze and new a thread to hear tweets from VSS continuously until obtaining entire tweets of specific things. During that, there is a progress bar to display the rate of this progress. If the progress bar keeps no change and the “Start” button stays inactive for a relatively long time, there might be input errors, offline issues or other problems. In this case, the program will pop out a time out message box if the IDE cannot receive tweets for 40 seconds. After time out, the button will change its text and function to exit and the user should click the exit button and restart the IDE. The user should check the internet connection and make sure to enter correct information.

For the hearing tweets part, the IDE only resolves tweets of each thing once in order to improve efficiency. Because the Atlas sends tweets in fixed orders, the IDE will remember the first tweet it receives from a thing. The next time it receives the same tweet, the IDE “thinks” that it has all the tweets of this thing and will ignore the announcing tweet from this thing.

The initialization may take long to several minutes due to the tweet announcing mechanism and is also influenced by the number of things.

After receiving enough tweets from the VSS, the program will write all information into the thing.csv, service.csv and relationship.csv respectively and wait for further usage. Then the user can click the “Start” button to start the IDE program and jump to the home page.

### 2.2 Home Page

The home page shows different buttons including “Things”, “Services”, “Relationships”, “Recipe” and “Apps”, each of which presents available things, offered services, possible relationships, app building editor and applications developed by IDE. Users can switch different tabs and jump to corresponding frames. The functions of these frames are described below.

In addition, there is a “Quit” button for the user to exit IDE.

### 2.3 Things

The Things tab shows the thing names and the ip addresses of the Smart things along with the picture of the RPI. Things tab is accomplished by storing data in the treeview where the data is read from thing.csv. A back button is placed in this tab and one can go back to the main page by clicking the back button.

### 2.4 Services

The services tab displays the services of each thing and images of services. There is a drop-down box at the top-right. Inside the drop-down box are the things’ ids. The titles of the services tab are “Service Name” and “Thing ID”. This part is achieved by reading data from “Service.csv” and using them to build a treeview of tkinter.

This tab has a filter function, which is implemented by the drop-down box, if a user chooses a thing id and the service table will update and show the services and images that are only related to the chosen thing. And if a user chooses “default”, then the service table will update and show all the service-thing-image.

### 2.5 Relationships

The Relationships tab displays the one-to-one relationships that exist in all services. It includes 4 columns: “Relationship Name”, “Relationship Type” (Control, Drive, Support, Extend, Contest, Interfere), “Service1” and “Service2”, which means a <Type> relationship called <Name> between <Service1> and <Service2>. This part is achieved by reading data from “Relationship.csv” and using them to build a treeview with tkinter.

This tab has a filter function which allows users to select the desired relationship type from the drop-down combobox. After selection, only records with the corresponding type will be displayed. Users can change the selected key words or cancel the filter at any time.

The tab also allows users to edit the relationships. It has a “create relationship” function. Users can click the “create new relationship” button and the IDE will display a new record without content. Then, users can double-click every column to edit the content of the new relationship. After filling the blanks, users can click the “ok” button to save the new relationship.

## 2.6 Recipe

The recipe tab contains two treeviews, two comboboxes and three buttons.

Treeviews:

1. The first treeview will show all the services that have been selected by the users to finalize an app.
2. The second treeview will show all the relationships that the user selects.

Comboboxes:

1. The first combobox will show all the available services read from service.csv and when the user clicks one item, the selected service will add automatically to the service treeview.
2. The second combobox will show all the available relationships read from relationship.csv, and will add the item to the treeview when the user clicks on that.

Buttons:

1. The first button is the finalize button, which is used for finalizing an app when the user has already selected all services and relationships that he/she needs. After clicking the finalize button, there will be a finalize\_app.txt file saved in the same path of the program. The finalize\_app.txt file will show all the names of service and relationship that the user has selected. After clicking the finalize button, the program will automatically clear all the services and the relationships in the treeview. At this point, the user can select the services and the relationships and finalize the app again, but the finalized file will cover the previous file (which means before the user clicking the save button in the app tab, the file will just only save the last version of your finalized app)
2. The second button is the clear button. clicking this button will clear all the services and relationships that the user has selected without saving into the file. After clicking the clear button, the user can select the items again to finalize the app.
3. The third button is the back button. The user can use this button to go back to the home page.

## 2.7 App

### 2.7.1 App Display

Click the app button and the window will show the app name and images that are related to each app. This function is implemented by reading file from "app.csv" and using the data to load the appname and the images. And there is a Scrollbar in this window, the "App manager" button will dynamically move with the Scrollbar, this is implemented by using relative position in placing the button.

### 2.7.2 App Management

- ◆ **Save:** After clicking the save button, the program will read the finalize\_app.txt file generated by the finalize button in recipe tab. Then read the service.csv to find the information of the services according to the service names in finalize\_app.txt. After finishing collecting all information, the program will generate all tweets for the services. And write the names of the services and the tweets with dictionary format into a txt file. Here the user is able to select the path and the name of the file by themselves.
- ◆ **Upload:** After clicking the "upload" button, the program will pop up a system window, which allows users to choose the app file. This window will first display the default "work directory" set by the program and all txt files in it, but the users can also manually select files in other paths to upload. After selection, the uploaded file's information will be written to "App.csv" and displayed in the App tab. Moreover, this function will automatically generate an icon (a black image) for the uploaded app. Users can change the icon in the "image" folder.
- ◆ **Active:** After clicking the active button, the program will turn the selected app into "active" state and open a new thread to begin to send tweets according to the app file. Tweets will be sent every 20 seconds according to the order of the services in the file until the user clicks the stop button or it has already run all services of this app. If we click the log button on the status panel, logs that are related to the specific app will show, the log will update about every five seconds, and after the log window pop up, the user cannot do anything unless he closes the log window.
  - **Status Panel:** After clicking the "status panel" button, the program will pop up a new window to display the status panel. The panel can show all running apps' names, their start time and their status ("active"). Clicking on any app's name button, the panel will pop up a window showing the logs of all the activity of this app until users close the log window (see "log" below for detail).In addition, there is a "stop" button in the top of the status panel. Users can manually stop a running app by clicking it (see "stop" below for details). When an app is stopped manually, its status will change from "active" to "inactive" and can still remain in the panel for 5 minutes. After 5 minutes, its status will turn to "removed", which means it will not be displayed when users open the status panel next time. For those apps that stop automatically, their status will turn to "Completed" after they stop. Then after 5 minutes these apps will be removed from the panel, like those apps that stop manually mentioned before.
  - **Log (invoked from status panel):** If the user clicks one of the apps in the status panel, the logs of the app will pop up a window showing the logs of all the activity of this app. The logs will update every 5.5 seconds. The log contents

are services with this app or “This app has stopped”. The user cannot do anything on the status panel until he or she closes the log window.

- ◆ **Stop (invoked from status panel):** In the status panel, after clicking the stop button, users can see a list of all running apps and can choose one of them to stop it. The thread of sending tweets will be killed and the app will turn to “inactive” state, and it will be removed from this tab after five minutes.
- ◆ **Delete:** After clicking the “delete” button, the IDE will pop up a system window which allows users to choose the app file. This window will first display the default “work directory” set by the IDE and all txt files in this directory. What’s more, the user can also manually select files in other paths to delete. After selection, the App file will be deleted from both local storage and the IDE. Also, its icon image will be deleted.

### 3. RESULTS

In this section, we will show how our IDE works and screenshots of the running process.

(1) At first, the Init page shows up and allows users to input information.

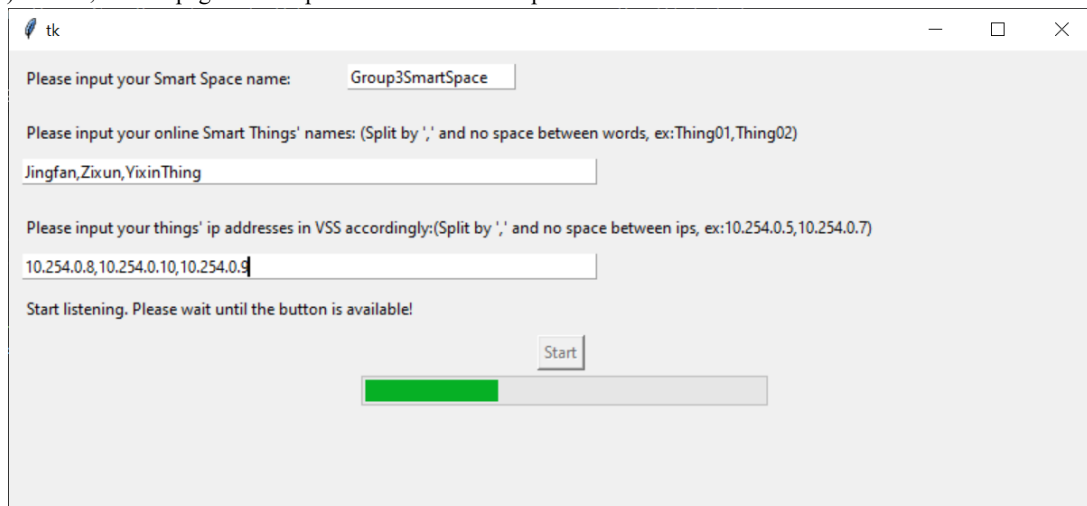


Figure 1

(2) If the input information is correct, jump to the Start page.

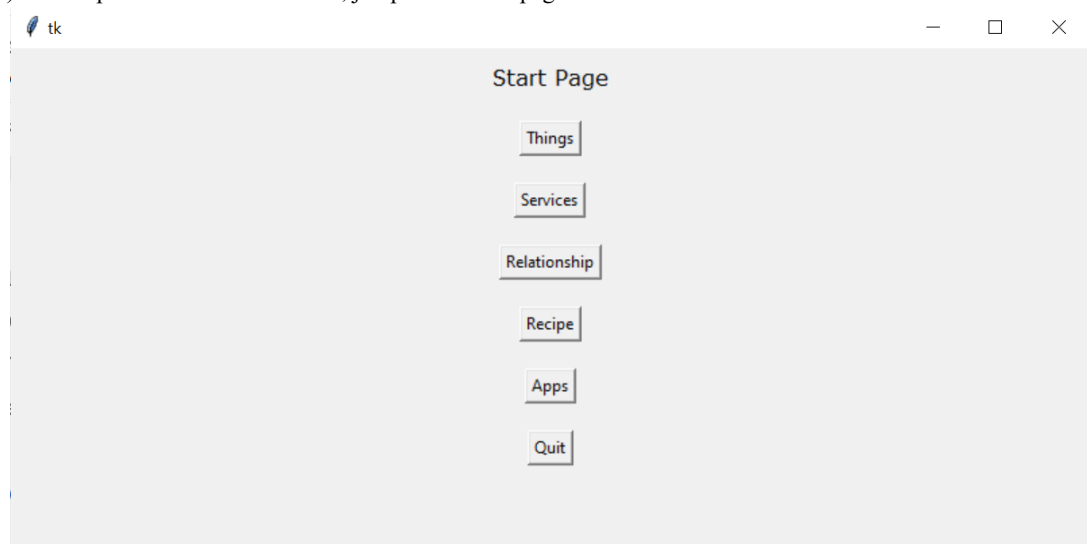


Figure 2

(3) Open “Things” tab.

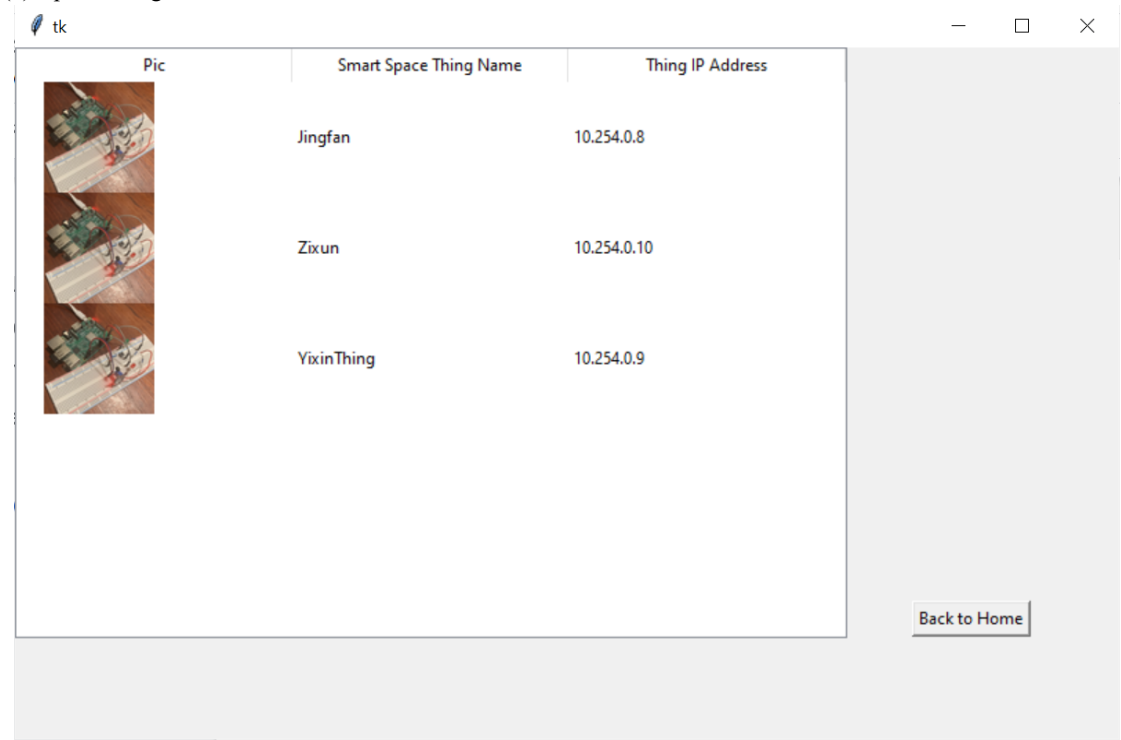


Figure 3

(4) Open “Services” tab.

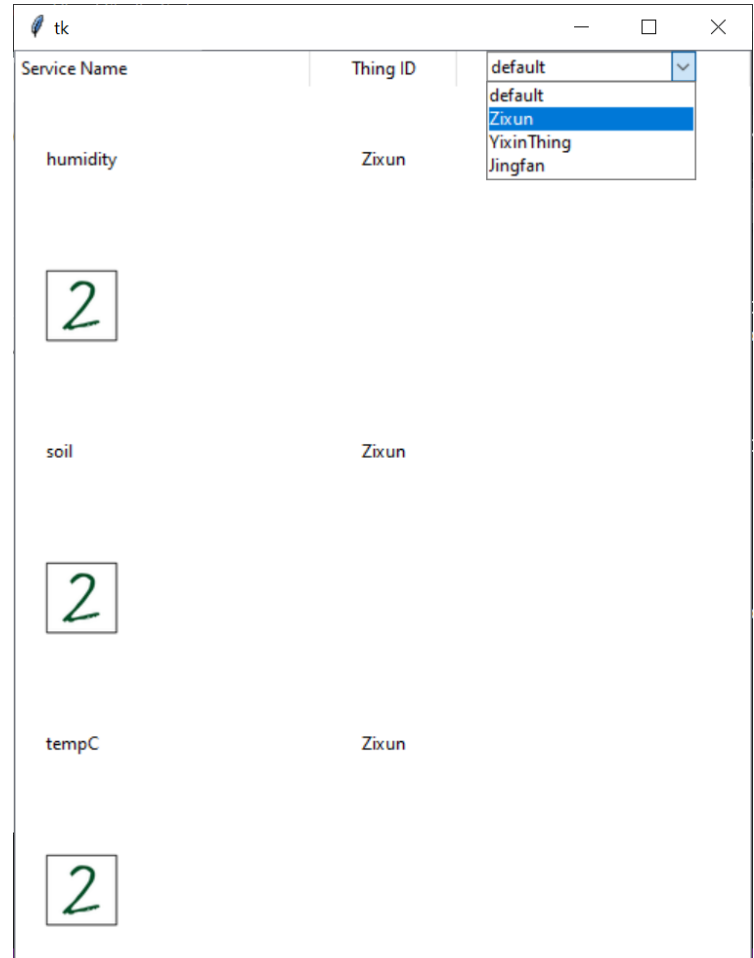


Figure 4

(5) Open “Relationships” tab. The frame can also show the existing relationships. (Because there are some bugs in Atlas that if we add relationships in the xml file, others will get aborted. So, in this page, there is no relationship read from Atlas.)

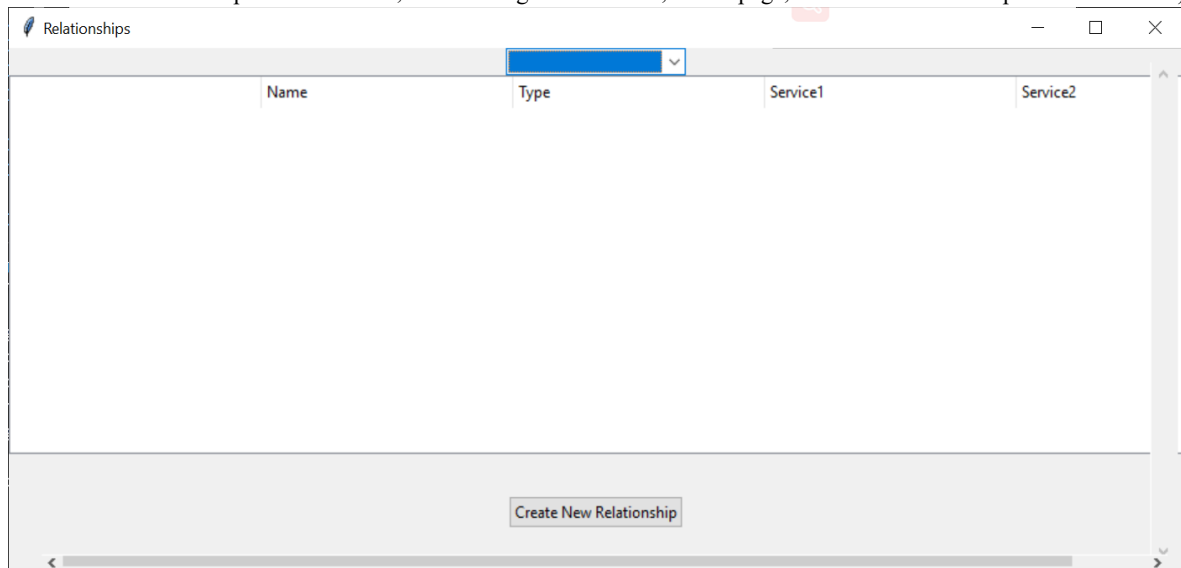


Figure 5

(6) Open “Recipe” tab. In this step, users can generate app building editors. Users can click “Finalize” when finishing editing.

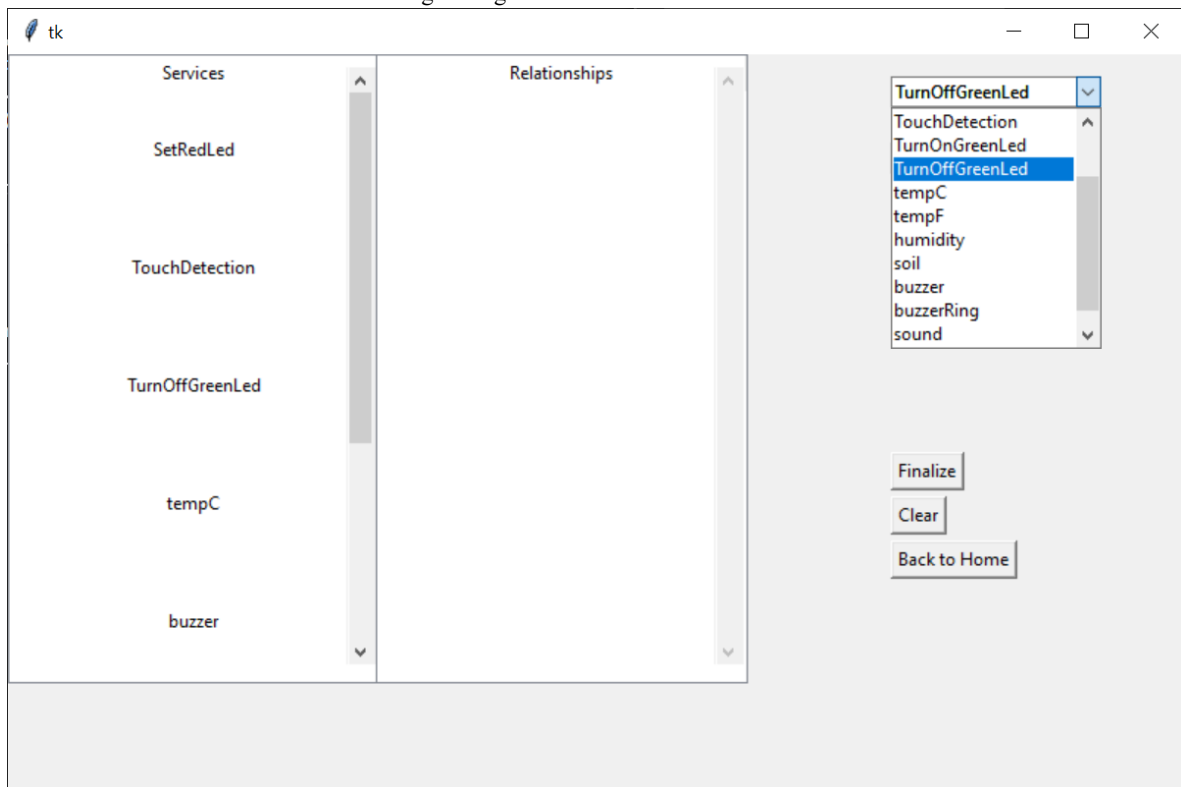


Figure 6

(7) After building recipes, users can go to the App frame.

a. Firstly, click “save” to save the finalized recipe into a text file. Note that you need to change to the development mode.

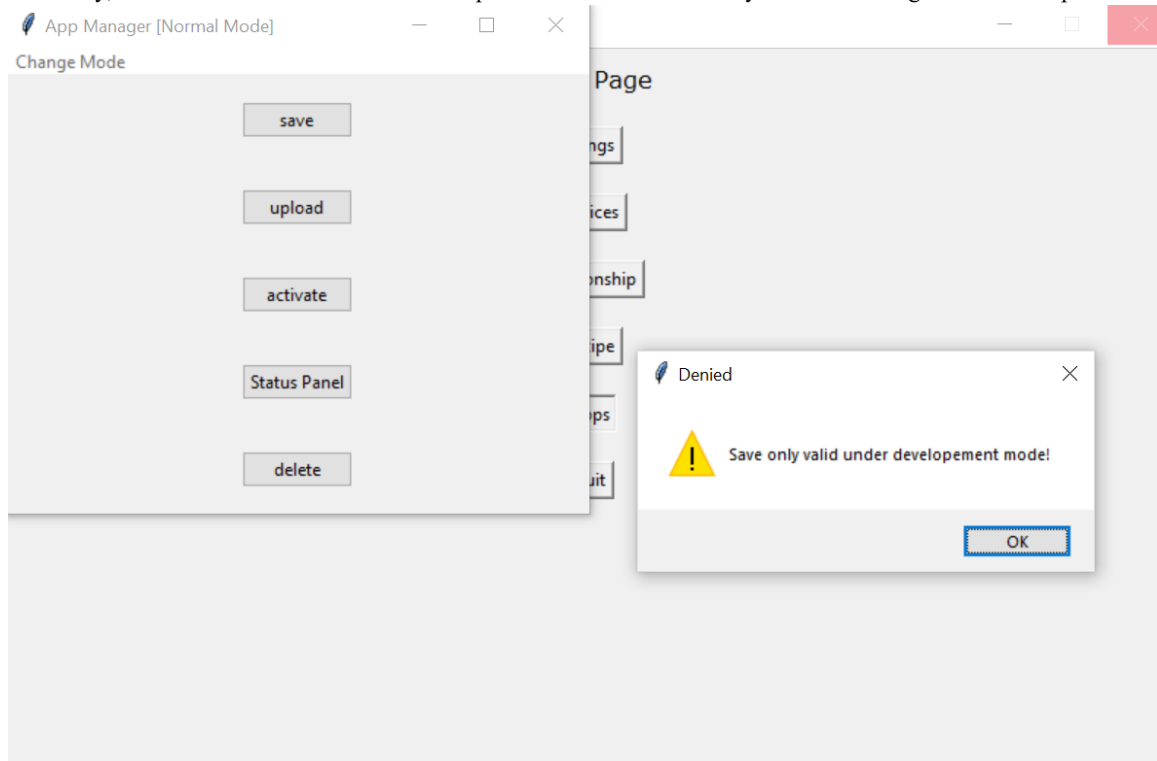


Figure 7

b. Users can also click “upload” to upload the text file containing a recipe into IDE.

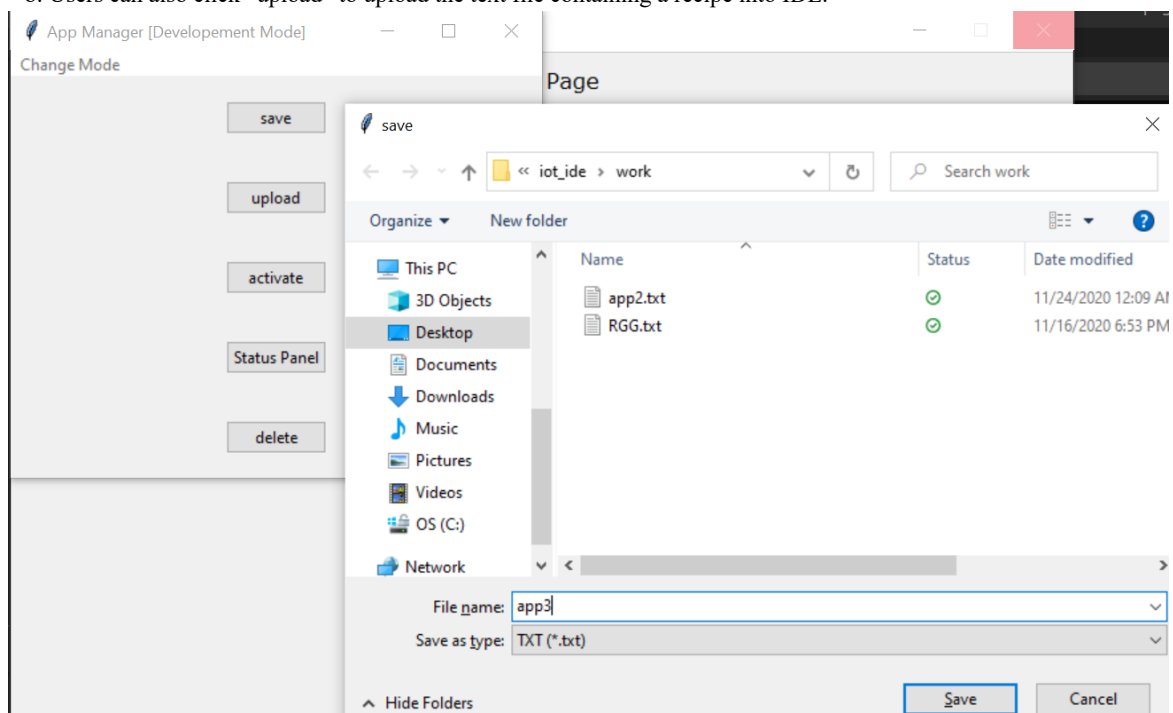


Figure 8

c. Click “active” to pop up a page in which users can choose one existing app to run.

tk (Not Responding)

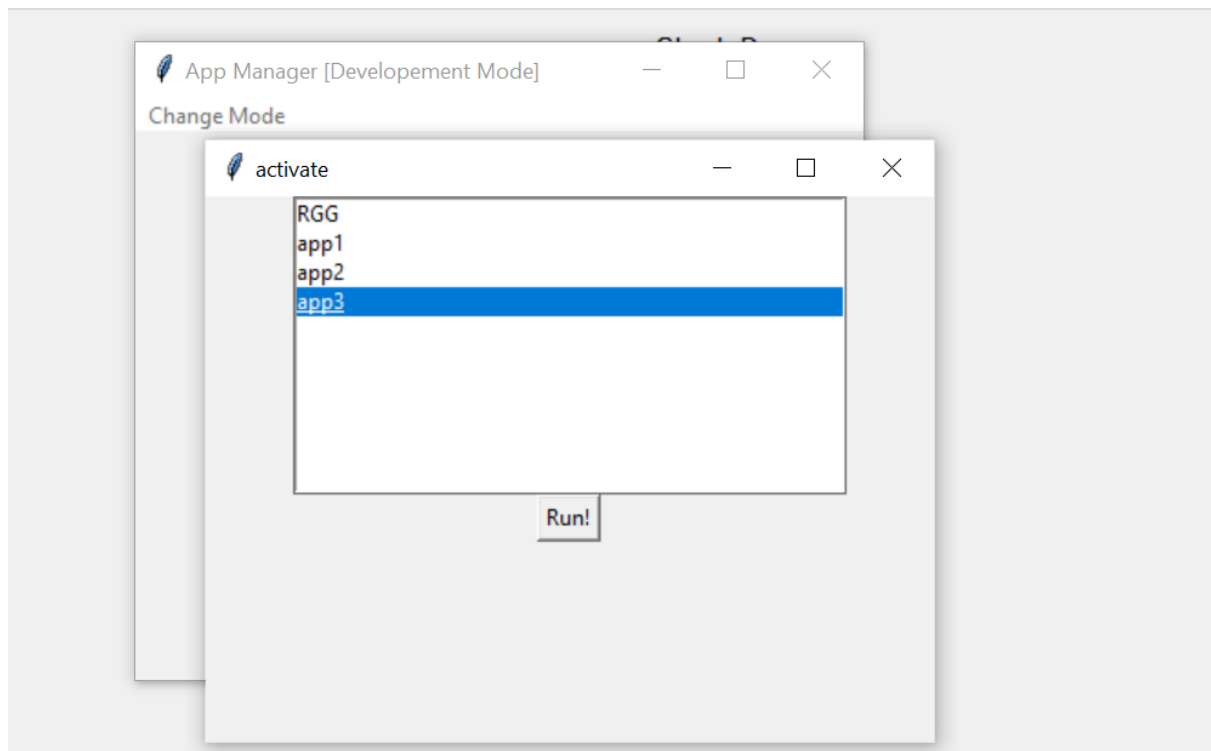


Figure 9

d. Click “status panel” to pop up a page showing the status of running apps.

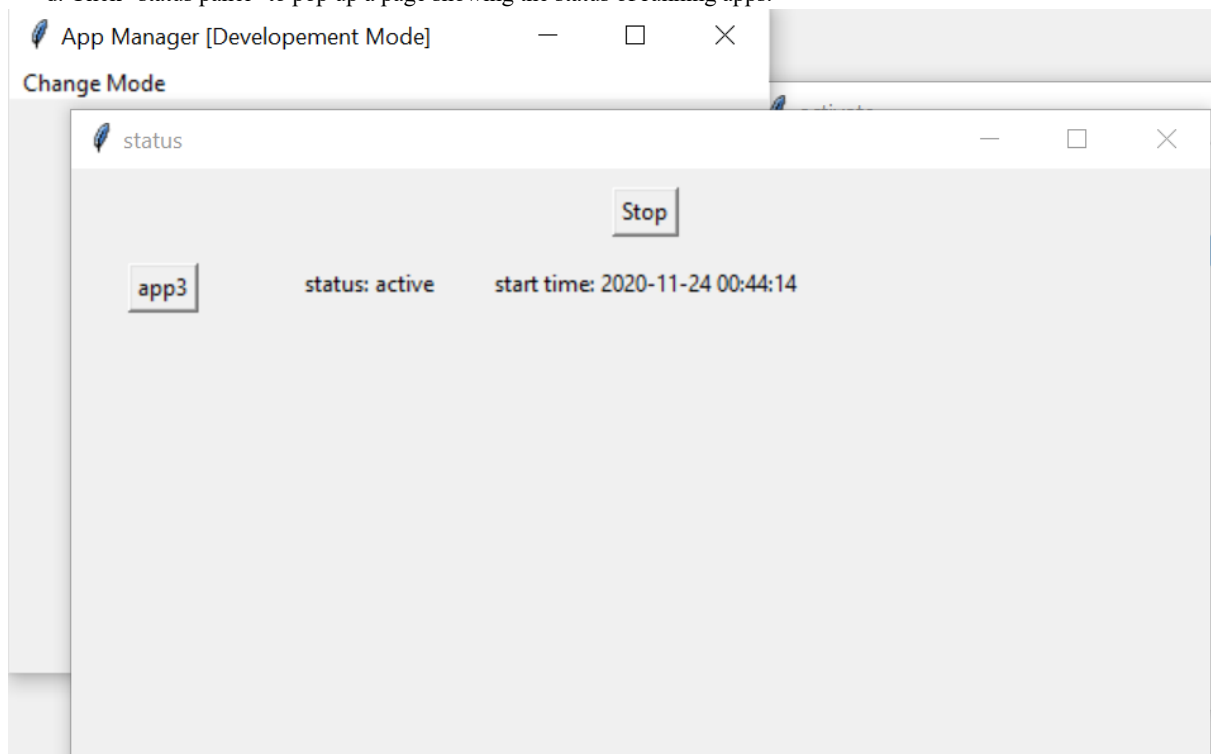


Figure 10



e. Click an app with the “active” status to show the log of this app.



Figure 11

f. Click “stop” to stop a selected app.

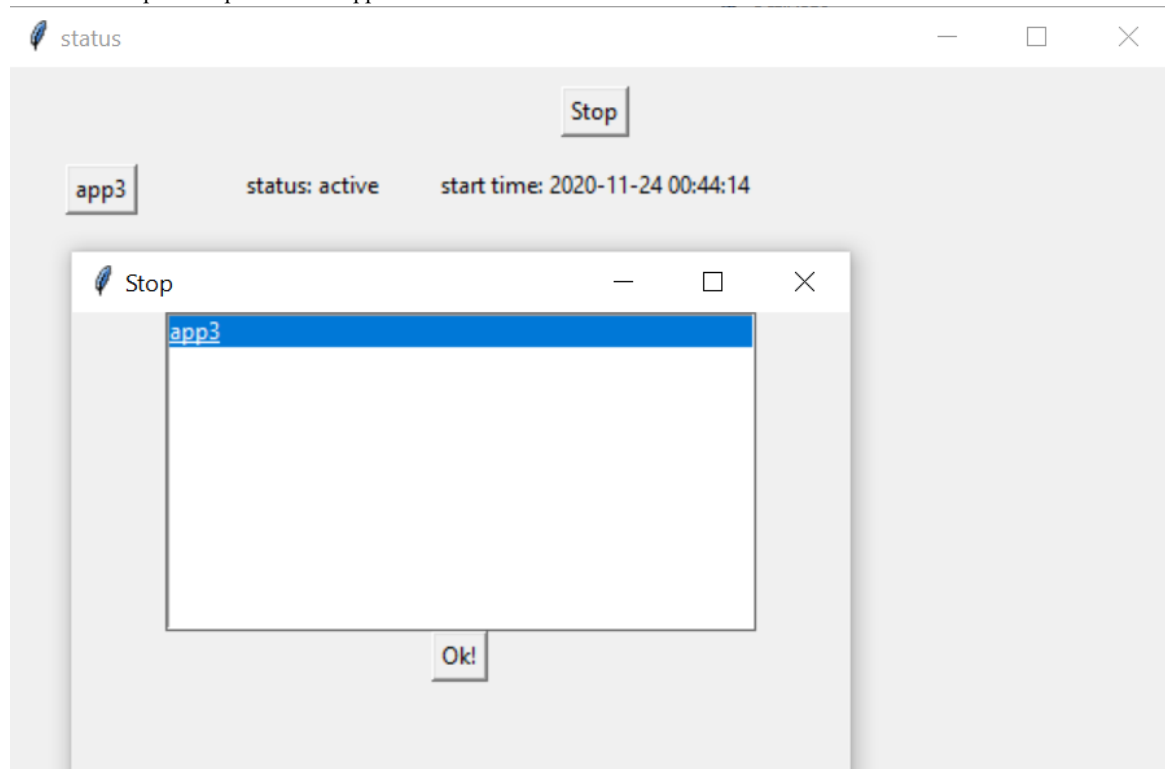


Figure 12

g. Now we can see this app has stopped.



Figure 13

Here are situations when a user writes invalid wrong inputs in the Initial Page. (Shown as Figure 14-18)

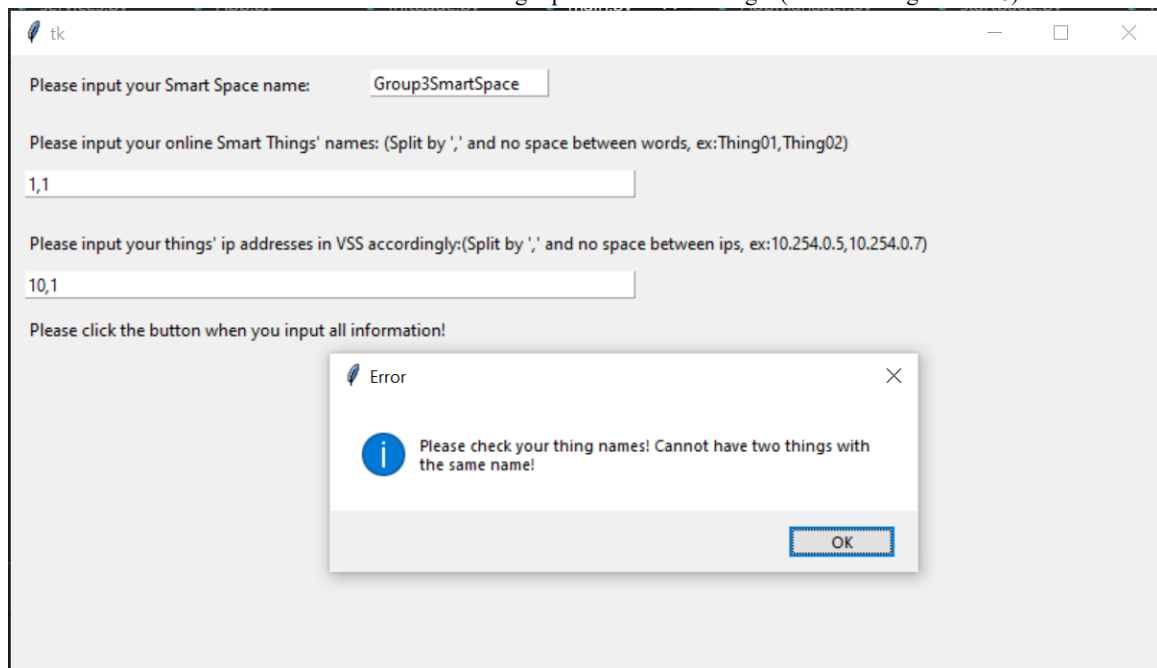


Figure 14

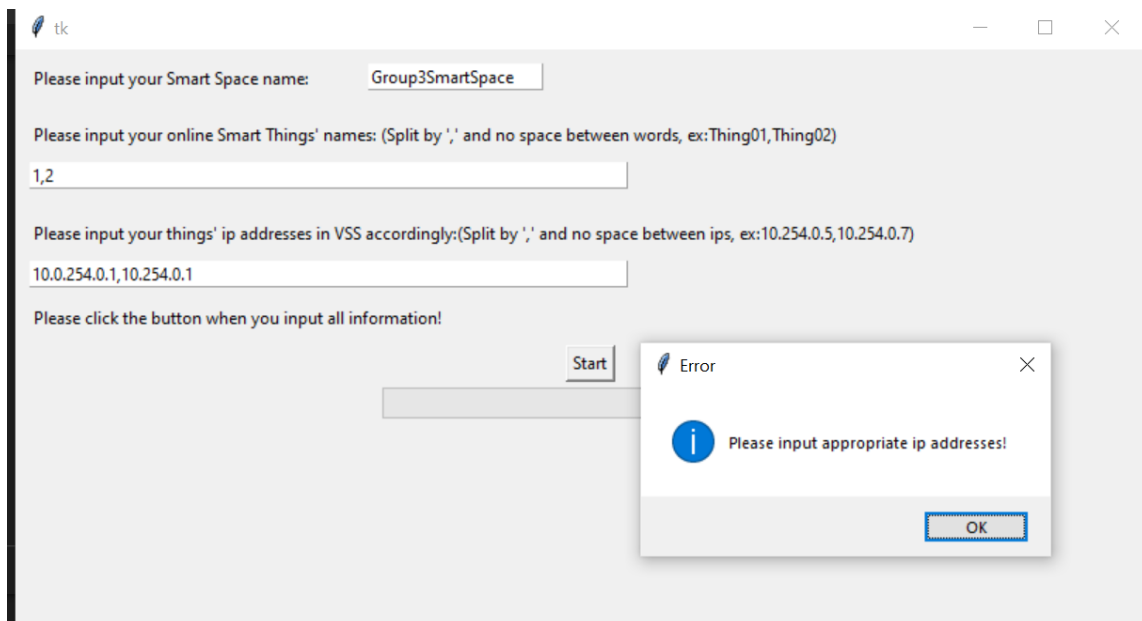


Figure 15

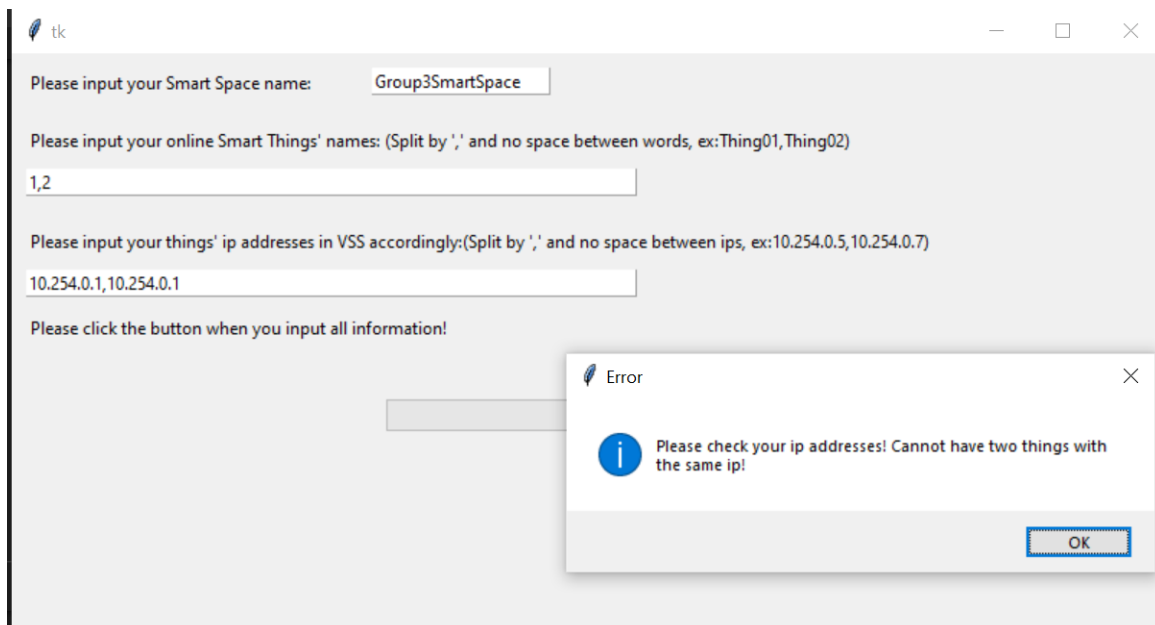


Figure 16

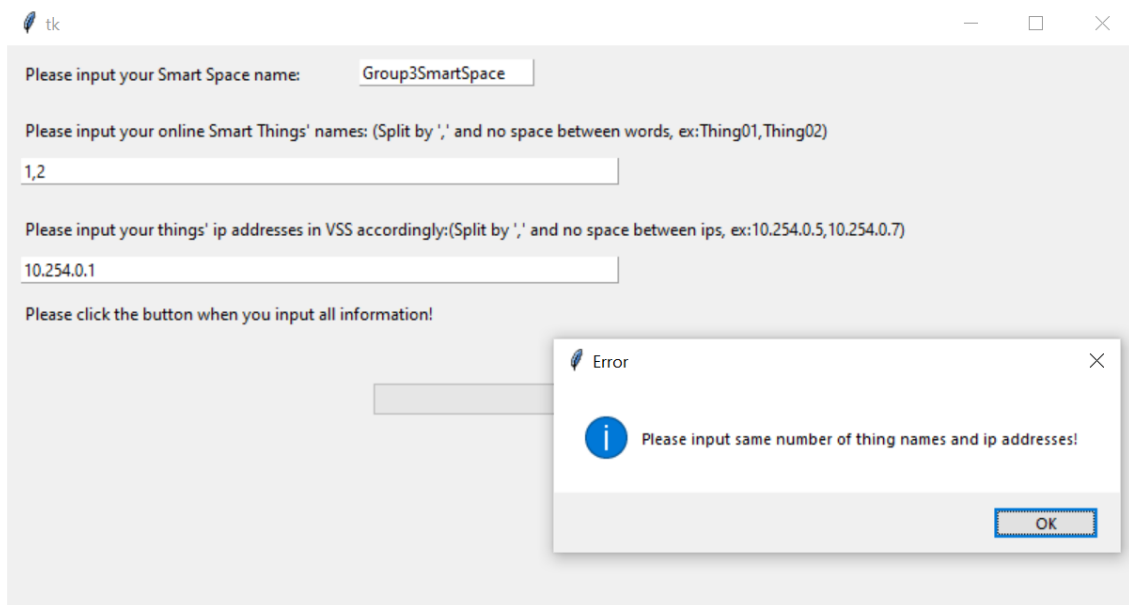


Figure 17

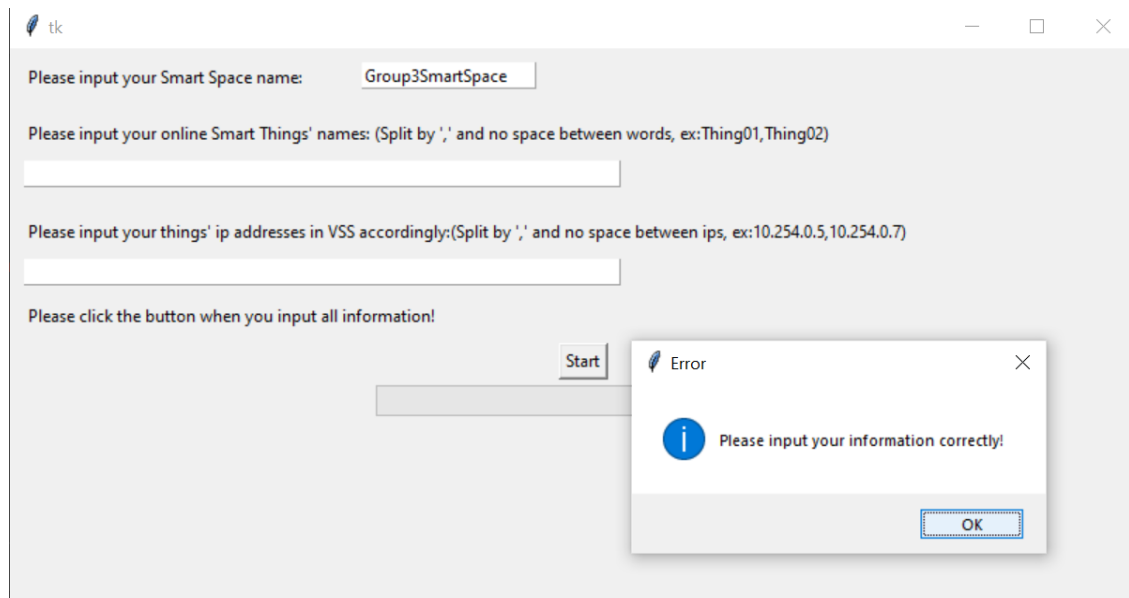


Figure 18

#### 4. USABILITY STUDY

To be continued in part 2 submission.

#### 5. CONCLUSION

To be continued in part 2 submission.