CS Dev Log

# Introduction

I will be using an agile development method based on versioning the app with iterations of the development cycle. For each iteration, there will be:

- Why is this version being coded?

- Requirments

- Pseudocode

- Test data

- Actual programming (The commented code will be available in the appendix)

- Testing

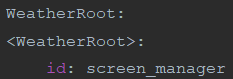
- Any changes needed

- Final Report

# Version 1.0

For version 1.0, everything up to actual programming is already in the report.

## Day 1 – Making the base appstate



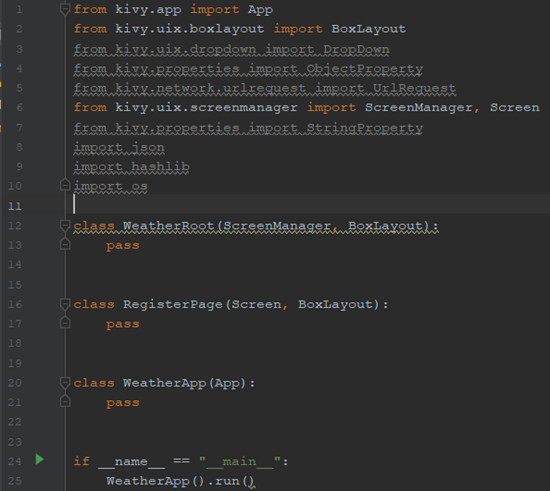
The classes have pass in them as we have not yet added any backend code into them. The classes also inherit from our different imports where needed for example, WeatherRoot inherits ScreenManager as it will be where the screen manager is so we can switch between different screens when needed.

The above sets up the screen manager for all our screens.

Here I have made the base app as well as imported what I’ll be needing for the app (can be seen in initial imports part).

It seems to pass our test so we will move o to our next part.

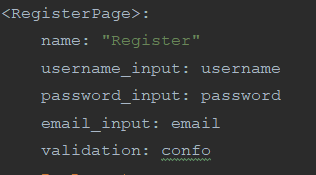
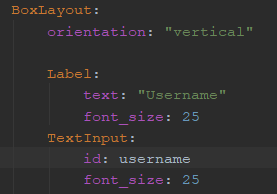
I have also set up the screen manager in the Kivy code for future use.



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | What are we testing for | Expected result | Test data | Success/failure + Proof | Notes |
| 1 | For a black window with Weather as the title |  | none | Success | Continue onto the next part |

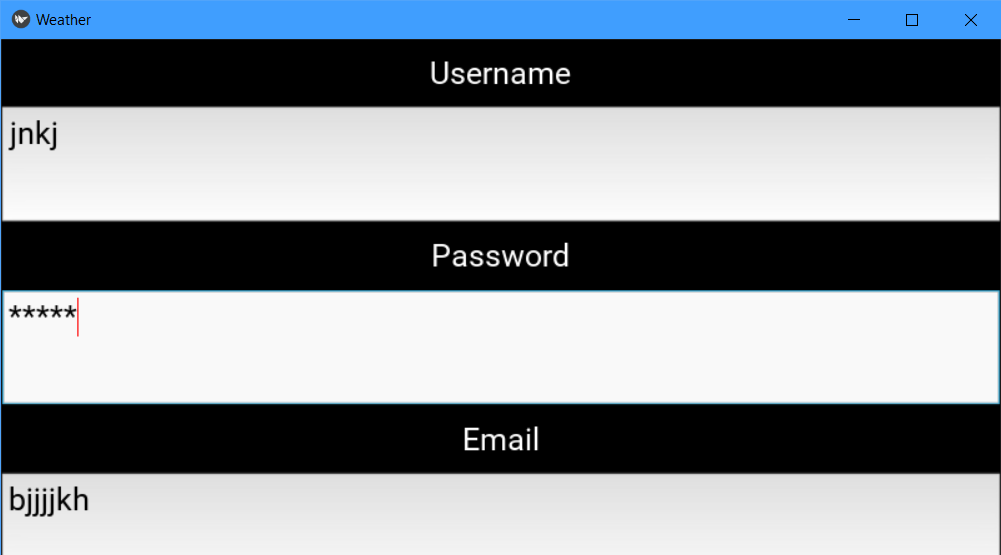
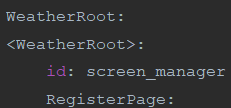
## Day 1, 2 and 3 – RegisterPage Class

### Day 1, making the frontend

The rest of day 1 consisted of preparing how the class would look to the user. We start with the buttons and labels.

I have created a vertically aligned box layout with 3 labels and 3 input boxes. The password input box has the password formatting added in. I have chosen a font size of 25 for ow to make the text clear and readable.

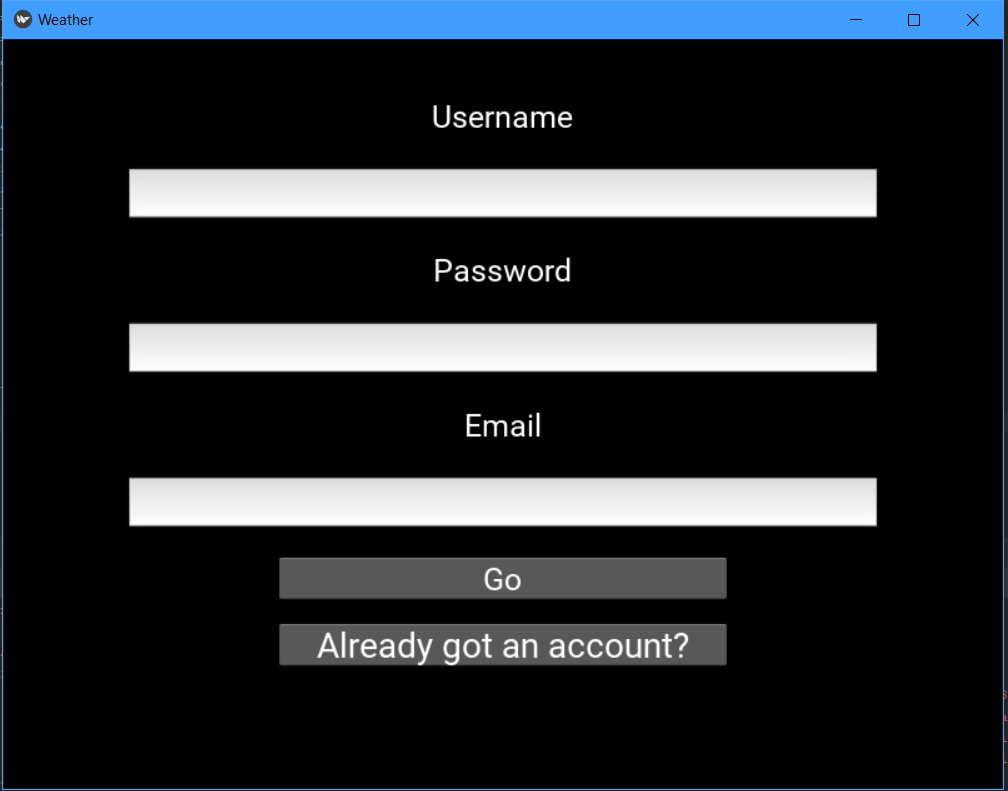
I have also prepared the object properties and the name for switching screens.

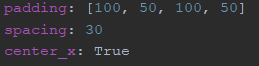


We ensure that this class is a child of the root/screen manager. This is so we can move between screens/classes in our app.

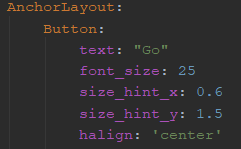
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | Check usage of the text boxes and to make sure nothing is missing | When input text into the boxes, it should be presented clearly and be correctly sized. | Just some random strings | Success (see screenshot) | none |
| 3 | To see if the password formatting works | Whatever text we put in the password box; it should be replaced with asterisks. | Random string | Success (see screenshot) | none |

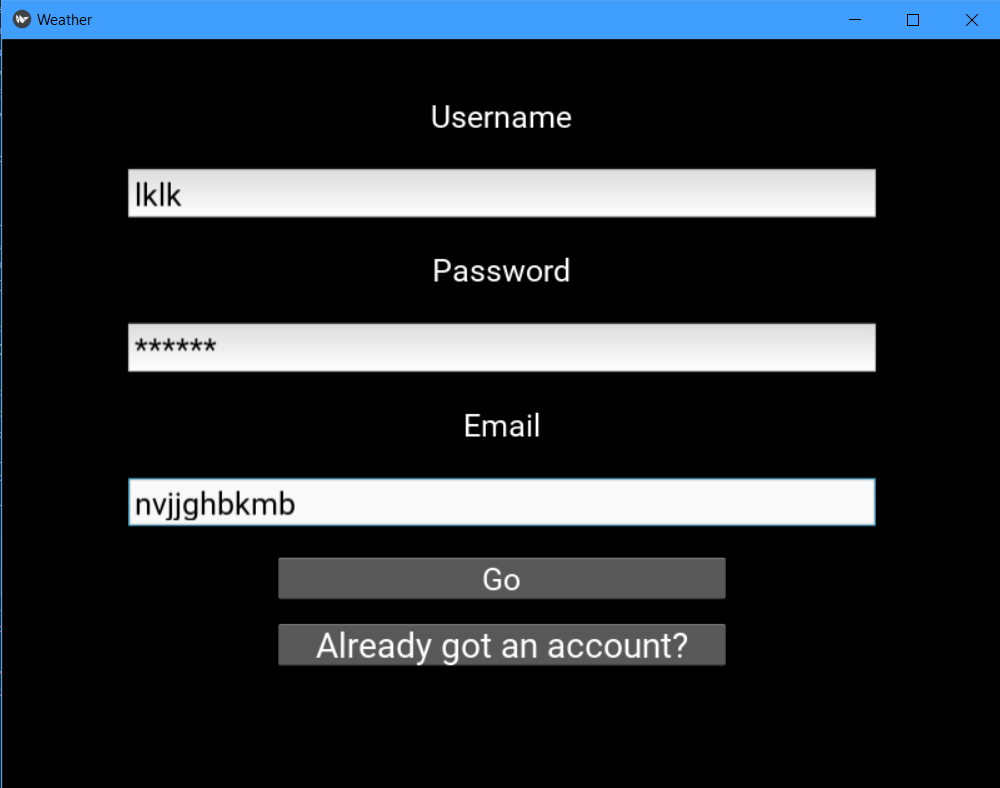
Next, I added padding and spacing to make sure the page looks presentable.





I have also placed the buttons in to make sure that the screen can handle all of the widgets on an already padded screen. The buttons were put inside anchor layouts and has the size hint changed to 60% width and 150% height.

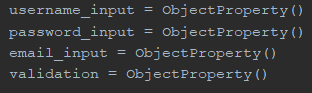




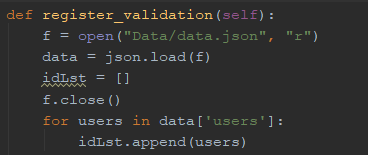
As you can see, the password input box is formatting the input with asterisks and the other inputs are clearly readable.

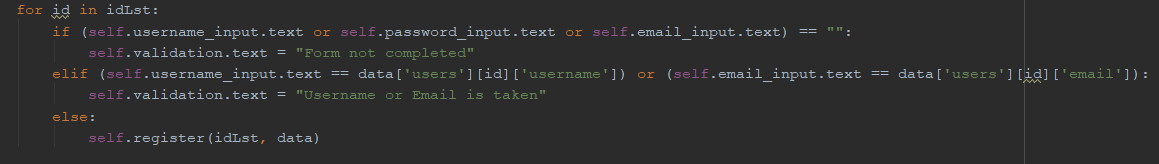
### Day 2 – making the backend.

To begin, I had to decide between using a JSON file or DB to store the data. After some contemplation, it was decided that a JSON file should be used now and a DB would be used depending on the success of the JSON file. So, following the pseudocode written, I began to write the code.



Our first job was to get the Object properties from the Kivy code with the help of the Kivy.properties.ObjectProperty class. This is so we can access the input box text and the confirmation texts.

Next, I decided to write the code for validating the register form inputs. This first included getting then formatting the data in the JSON file so we can iterate over it. The file has been closed sooner than later to save system resources.

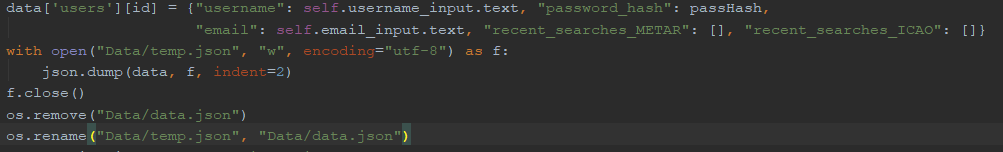


Here we validate that the user has completed the form as well as iterating over the data to check if the user already exists in the JSON file. If they are then the confirmation text should change based on this.

A new id is created for the new user and hash for their password is made using md5 encryption and salt to protect against hackers and the salt to protect against rainbow tables.







Lastly, the program updates the extracted data with the new user’s data (id, username, password, email, etc.), in the correct formatting. Then the json file will be closed, then this data will be dumped to a new temporary file, the old one deleted and the temporary file renamed to the old one’s name using the os module.

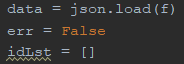
Finally, to conclude the day, I decided to run some tests. The results can be seen below.

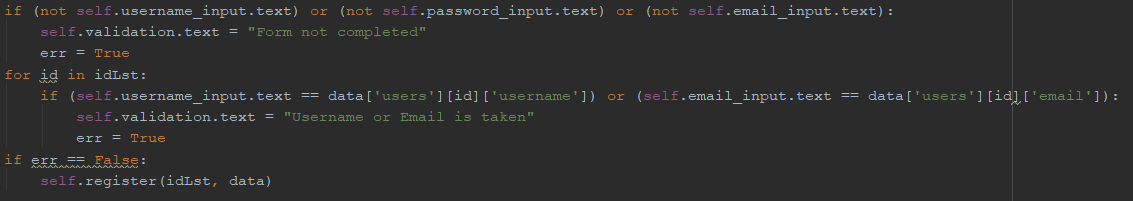
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | What are we testing for | Expected result | Test data | Results | Notes |
| 4 | Test validation process, no username | Expect to see “Form not completed” show up in the confirmation text area and for no registration to continue. | Random strings entered for password and email, nothing in the username input box. | Failure, even though it says form not complete:    The registration process still continues. | We will need an overhaul of the validation system. See below for how I fixed it on day 3. |
| 5 | Test validation process, no password | ^ | Random strings entered for username and email, noting in the password input box. | ^ | ^ |
| 6 | Test validation process, no email | ^ | Random strings entered for password and username, noting in the email input box. | ^ | ^ |
| 7 | Test validation, already existing user | “Username or email is already taken” warning message. | Username = “f”, password and email have random strings | Failure, it just sends the data for registration anyway, confirmation text says “Form not completed”. | Need to redo on day 3. |
| 8 | Testing validation, existing email | ^ | Email is “[h@h.h](mailto:h@h.h)”, password and username are random strings. | ^ | ^ |
| 9 | Test with legitimate data which hasn’t been used | It should register the user and there should be a confirmation message for the user. | Username = “re”  Password = random string  Email = “re@re.re” | Failure, it does send the data off perfectly but the confirmation text still says that the form is not complete. | ^ |

### Day 3 – Fixing the validation system

To start off, I tried to diagnose where the problem was coming from so I had to add some print statement’s in the code to see how it is running in the terminal. This didn’t work as all I could get to print out were the inputs and existing data so I decided to look at the code again.

Here I realised that the bit which checks if the form is full is stuck in a loop even though there is no need for this. There is also no way to break out of the procedure in case of an error. So, I devised a plan where there could be a Boolean which is set to false, whenever there is an error, it is set to true. At the end of the procedure, if the Boolean is false then it will go to the register procedure, otherwise, the backend will stop.

For the code, I have called the Boolean err.  
err is changed to false when a validation error occurs and is validated at the end asking if it is false.



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | Test validation, no username | Form not completed message and no updated JSON file | Random strings entered for password and email, nothing in the username input box. | Success |  |
| 11 | Test validation, no password | ^ | Random strings entered for username and email, nothing in the password input box. | Success |  |
| 12 | Test validation, no email | Username or email is taken message and no updated JSON file. | Random strings entered for password and username, noting in the email input box. | Success |  |
| 13 | Test validation, already existing user | ^ | Username = “f”, password and email have random strings | Success |  |
| 14 | Test validation, already existing email | ^ | Email is “[h@h.h](mailto:h@h.h)”, password and username are random strings. | Success |  |
| 15 | Test with legitimate data. | Should confirm that registration is complete | Username = “re”  Password = random string  Email = “re@re.re” | Success |  |

### Checking with our criteria

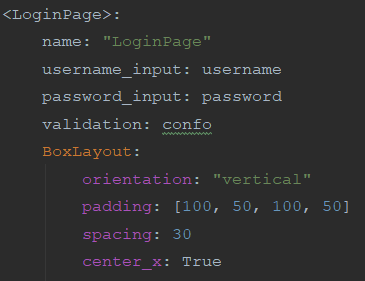
|  |  |  |  |
| --- | --- | --- | --- |
| Criteria | How to check | Completed with tests successful (Y/N) | Notes |
| A register system | We are hoping for a working register system that can add new users and give error messages based on inputs. | Y |  |
| Appropriate validation of the inputs | Test it with all sorts of inputs e.g. missing inputs, existing users, etc. | Y |  |
| The data to be kept secure e.g.: hashing passwords with salt. | The passwords should be hashed with salt where the data is stored. | Y |  |
| The user’s data is set up in the correct format (TBD) | Check where the data is stored that new user’s data is in the correct format | Y |  |
| Error messages if unsuccessful | Check by putting in data that should return an error message and check if the error message has appeared. | Y |  |
| Success message | ^ but with data that would return a success message. | Y |  |

Now onto the next class.

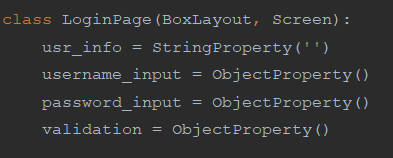
## Day 4, 5 and 6 LoginPage Class

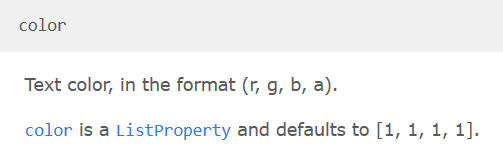
### Day 4 - Making the frontend

After the relief from yesterday’s work, I set out on making the frontend for 1.2.

I started with the basic design for the boxlayout with the padding which had worked with 1.1

Of course, this would not display anything just yet, but it would help with structuring our widgets in a reasonable manner on the screen.

  
We also initialise the class in the python script (otherwise, the root would raise an error). It inherits from boxlayout so we can use it in the backend and Screen so we can connect it to the screen manager system

With the login page, I decided that I wanted to add a bit of colour to the font so I researched how the colour system works and got this.

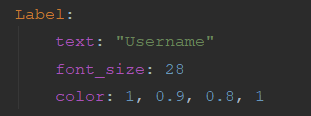
Colour in Kivy, is between 0 and 1 (with allowance for 1 decimal place) for all the rgba values so the default (1,1,1,1) is black with no transparency. As I wanted to add my favourite colour for text on a black background, blanched almond, I had to think of a way to turn the 0 – 255 system into the Kivy system.

Blanched almond in the 0-255 system is

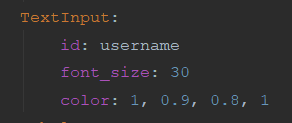
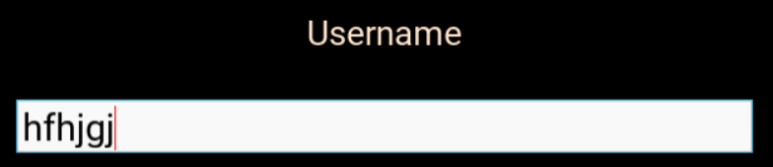
So how about we divide these values by 255 and round to 1 decimal place. Then we get:

1, 0.9, 0.8, 1 (1 at the end as we want no transparency).

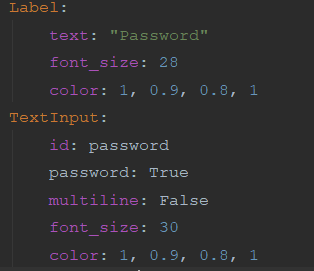
Let’s try it out then:

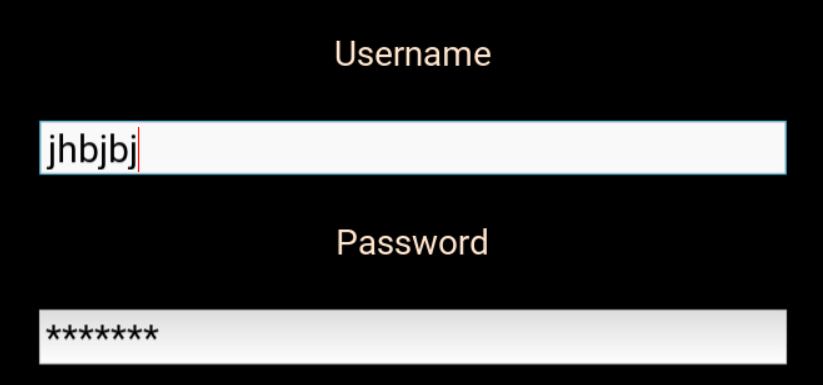


Well that went well, how about we try with a text input

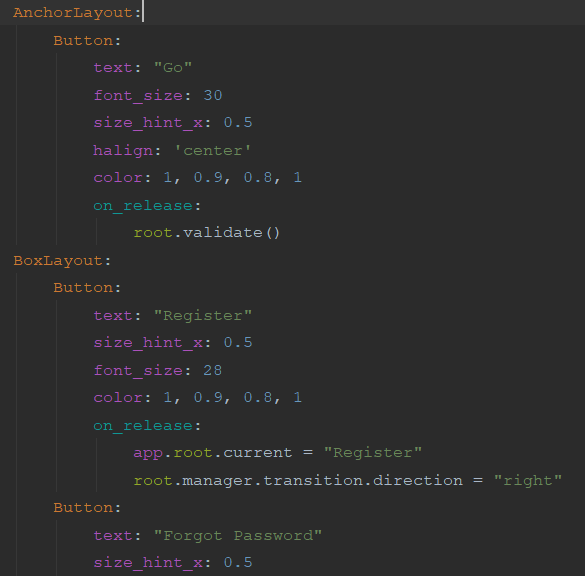


I guess it didn’t work. How about we try colouring a button's text?

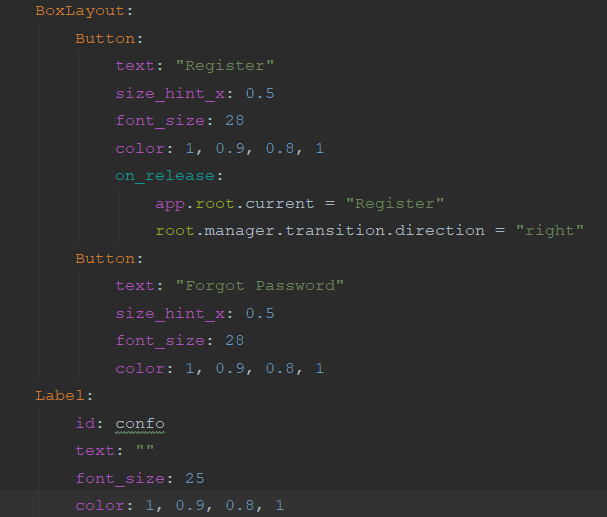
Before we do that, I think we should add our other input and label in – for the password.



Now that that works, we can go back to experimenting with the colouring as we will add the submit button next.

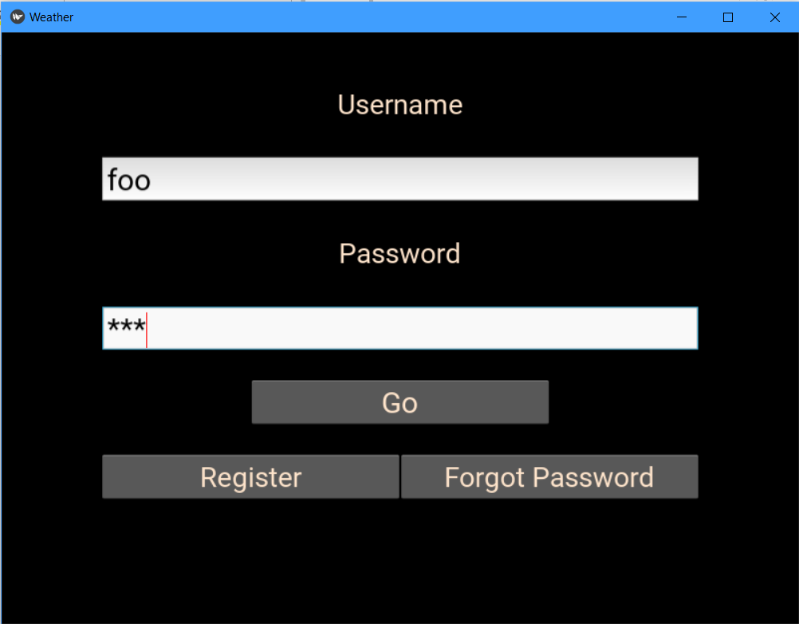


Great. Now we might as well finish off the frontend with the last 2 buttons in a box layout and the confirmation text.

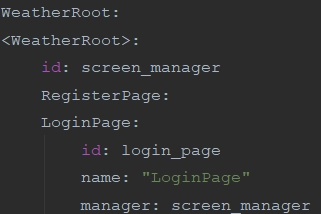


Now that those work, we can run our tests just for procedure’s sake

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | What are we testing for | Expected result | Test data | Results | Notes |
| 16 | Check usage of the text boxes and to make sure nothing is missing | When input text into the boxes, it should be presented clearly and be correctly sized. | Just some random strings | Success (see screenshot) | none |
| 17 | To see if the password formatting works | Whatever text we put in the password box; it should be replaced with asterisks. | Random string | Success (see screenshot) | none |

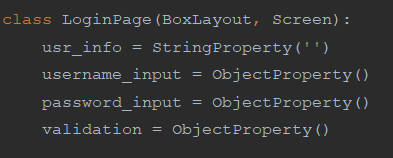


Now that that’s that, let’s move on to the backend of 1.2. I’m sure you realise why blanched almond was a great choice now.

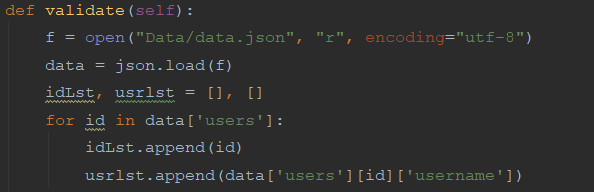


We make this class a child of the root/screen manager, we identify this clearly in the last line.

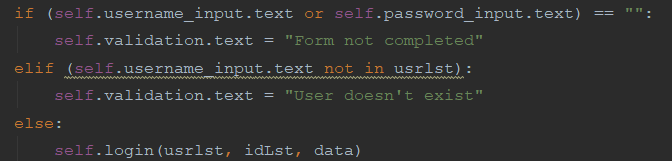
### Day 5 and 6 – The Backend

Now it’s time for the backend code. Hopefully, I don’t have any issues this time.

We begin by bringing the object properties (well more like establishing a link between the 2) from the frontend and saving them as variables.



Now we should begin with the validation procedure. For this, I loaded up the JSON file, formatted it and set the user's data and usernames as separate lists. This is so we can check if their username exists and we keep the user’s data to confirm their password in the next procedure.



This validates for a completed form and if the user exists. If it fails then an error message is set as the validation text. If it passes then it moves to the login procedure with the data and the lists as arguments.

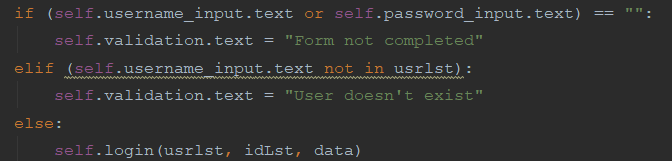
Testing:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | What are we testing for | Expected result | Test data | Results | Notes |
| 18 | Checking if the validation which checks if the form is filled woks. (no username) | The validation text should change to “Form not completed” | Nothing as username as a random string as password | Failure | This will be fixed after the tests are done. Also, the validation text looks good in blanched almond, right? |
| 19 | ^ but with no password | ^ | Nothing as password but some random string as username. | Failure | ^ |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | What are we testing for | Expected result | Test data | Results | Notes |
| 20 | ^ but with no username or password | Same as 19 and 20 | Nothing for either username or password | Success | none |
| 21 | Test if the validation to check if the user exists | A surprise | A user that exists for the username e.g. f and some random string for the password | Success  Well we were expecting that error, right? As we haven’t created the login method yet. | Add a login method pls |

To end day 5, let’s try to fix the errors.

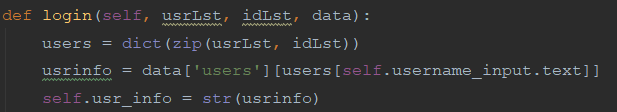
After a quick consolidation session with Quackius V of Mancunium (my rubber duck), we found out that I was dumb and it was quite an easy fix. We replace this:



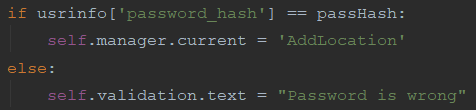
With this:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | What are we testing for | Expected result | Test data | Results | Notes |
| 22 | Checking if the validation which checks if the form is filled woks. (no username) | The validation text should change to “Form not completed” | Nothing as username as a random string as password | Success |  |
| 23 | ^ but with no password | ^ | Nothing as password but some random string as username. | Success | ^ |
| 24 | ^ but with no username or password | Same as 22 and 23 | Nothing for either username or password | Success | none |

That took up all of day 5.

Day 6

We begin the next procedure creating a dictionary from the 2 lists carried along so it is easier to find the user’s info. Then we load up the specific user’s data with the help of dictionary querying and the newly created dictionary. Finally, we set this as our global variable so we can use it later.

Here we set a salt and we create out hashed password using our salt (and encoding it for integrity).

Finally, we check if the hashed input password = the one stored in the database. If it matches then the user is moved to the main class (which is the metar searcher – poorly named as ‘AddLocation’ using Kivy’ s screen manager. If it fails then an error message is shown to the user via the validation text.

Time for some testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | What are we testing for | Expected result | Test data | Results | Notes |
| 18 | Checking if the validation works | The validation text should change to “Password is wrong” | Nothing as username and a random string as the password | Success | The password is actually f |
| 19 | ^ | There should be an error message as we have not created the AddLocation class yet. | Nothing as password but some random string as username. | Success | Gr8 success, we can move on to the next class now. |

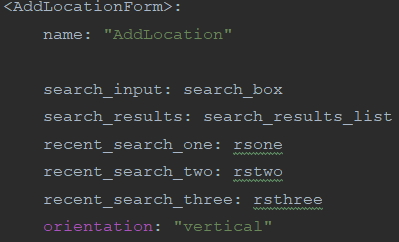
### Checking with our criteria

|  |  |  |  |
| --- | --- | --- | --- |
| Criteria | How to check | Successful with testing done (Y/N) | Notes |
| A login system | If there is a working login system with good validation which takes the user to the main screen when successful. | Y |  |
| Appropriate validation of the inputs | Fire in some inputs which should not make the user log in e.g. no password filled or the username not existing | Y |  |
| Matching the hashes | We need to check that when we input the correct password, it should log in which shows that the hashes have been matched. | Y |  |
| Error messages if unsuccessful | Fire in some bad data, if we get an error message then we’re cool. | Y |  |
| On success, go to the main screen (TBD) | Fire in some correct data (can be done at same time as matching the hashes) if the main screen loads then we’re Gucci | Y |  |
| On success also save their data as a global variable until the program shuts down. | This we can only check once we have linked this data to something else e.g. a recent search system if they load properly then it works. | Y |  |

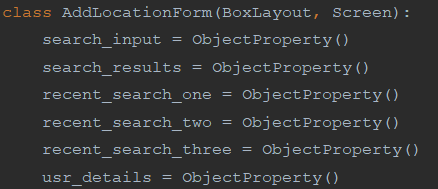
Now onto 1.3

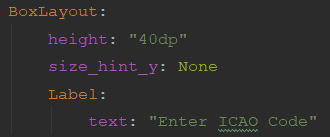
## Day 7, 8 and 9: AddLocationForm class

### Day 7 – The Frontend

🡨 Firstly, we make our class a child of the root/ screen manager so we can move along classes smoothly. We also make sure that we can access the user info by setting it as an object property of our new class.

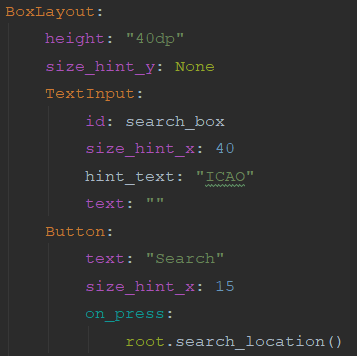
Next, we set up the class itself and its object properties and its root settings 🡪

Then we set up the class in the python code remembering to inherit from Boxlayout and Screen.



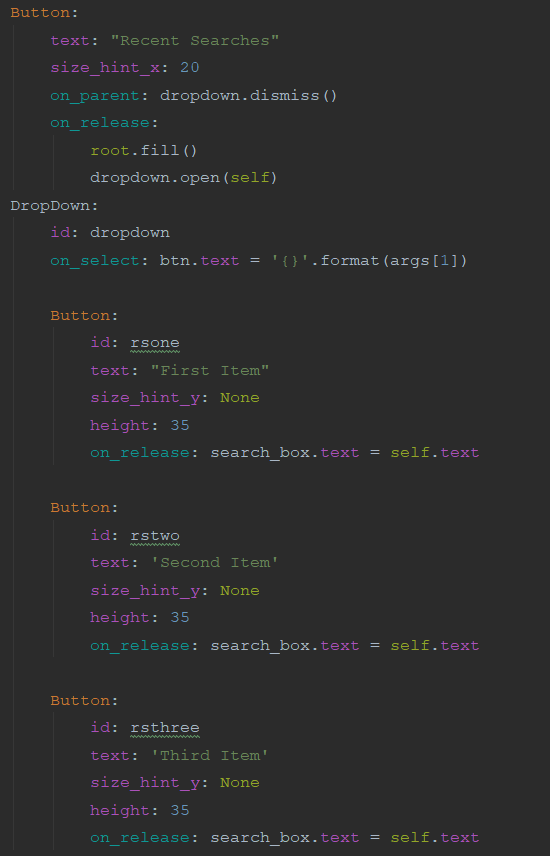
We set up the title text (following our pseudocode properties of the widget) and run a test to make sure everything is working

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | What are we testing for | Expected result | Test data | Results | Notes |
| 20 | If we can access the class directly after login | The class should load and we should be able to see the title text. | Just putting in an existing user’s info e.g. Username = h, Password = h. | Success |  |
| 21 | If the title text appears and is the correct size | The title text should appear at the top and be of font size 40 | ^ | Success (see above) | Surprisingly, the text auto-aligns to the centre. Guess that’s less work for the next versions. |

Here, I create a new box layout for the rest of the widgets (except the ListView) which conveniently auto aligns the widgets horizontally so we can use size\_hint\_x to specify the width of each of these widgets (as long as they all add up to 100).

The id is added to the input box so we can capture the search. We link the button to our currently non-existent backend code.

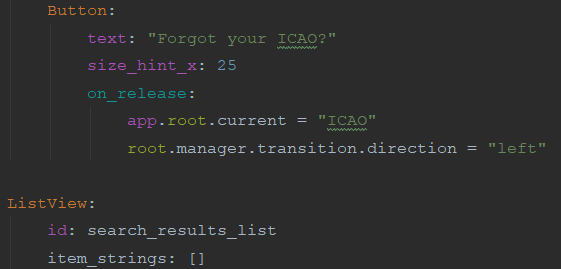
The search box has to be larger than the button as it needs more space for inputs and naturally is larger in most designs.

On to the recent search system. First, we create the button which is set to fill the dropdown buttons then to open the dropdown when clicked. These dropdown buttons can then be clicked to make the search box have the same value.

Next, we create the buttons and id them so we can access the value.

Finally, we see them up and add the on\_release code.

It is basically the same thing for all 3 buttons apart from differences in id name and text.

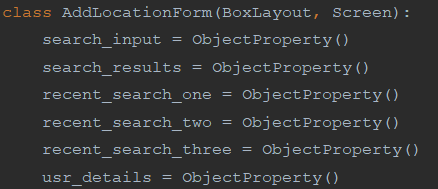


(Above left), we finish off the backend with the button which links to the ICAOFinder class and the list view for the search results. Now for some testing.

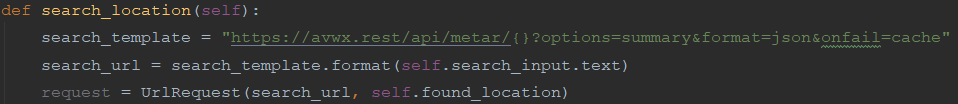
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | What are we testing for | Expected result | Test data | Results | Notes |
| 22 | If the buttons load up | The widgets should be loaded horizontally, with correct sizing | none | Success |  |
| 23 | If the dropdown works | It should open up, revealing 3 buttons that should be stacked vertically underneath the dropdown button. | none | Success |  |

We will test the buttons in conjunction with the backend code later.

### Day 8 – The Backend

We start the day by initialising the object properties (usr\_details is the one which we are porting from the login page).

I have changed the usr\_details call to an ObjectProperty instead of a StringProperty here as it makes no difference as it calls in the same data and fits in with the other calls.

Next, we set up the search location class which prepares the API request and sends it off. When it returns successfully it should move onto our next class.



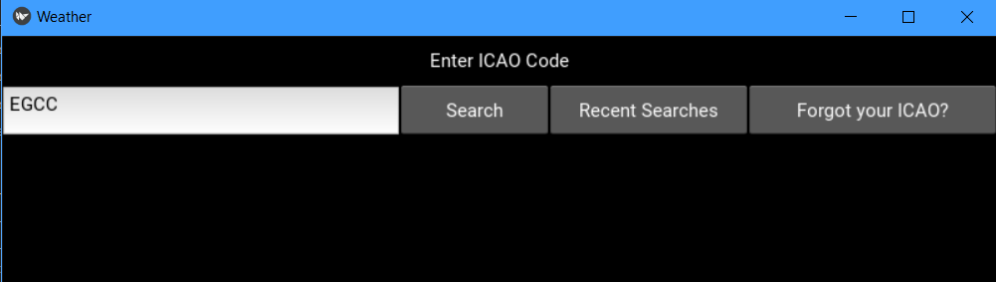
For now, I’ll have a little message show in the console to confirm that the request has been completed successfully.

Now to test it

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | What are we testing for | Expected result | Test data | Results | Notes |
| 24 | Check if we can access the API successfully. | We should get a message on the console. | EGLL as the ICAO | Failure  There seems to be no message in the console after waiting for quite a while… | Guess I’ll have to diagnose the bug. |

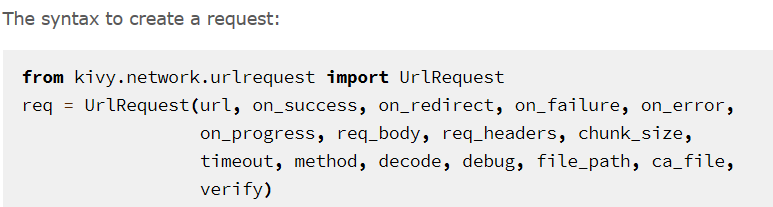
Finding the problem

Firstly, I should reset my WIFI and check if it works



Guess not…

Let’s check the docs again and see if there is anything that can help us there.



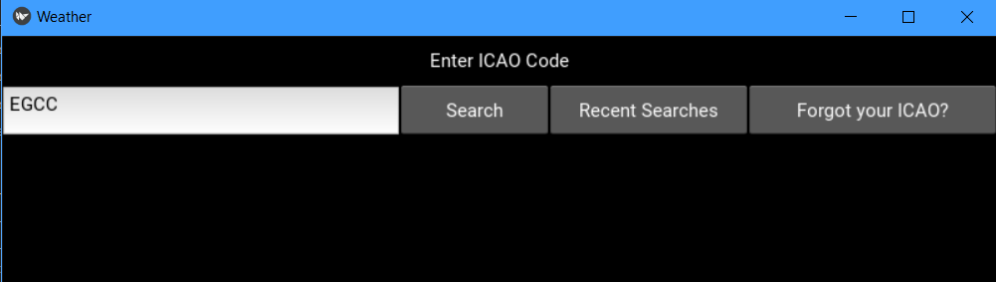
It seems like the ide may have not understood the parameters properly so how about we add the URL and on\_success parameters in? So, it changes from this:



To this:



It seems not to fix the error:



I guess we can try the on\_error parameter. We’ll make it print something if there is an error. So, we should change it from:



To:

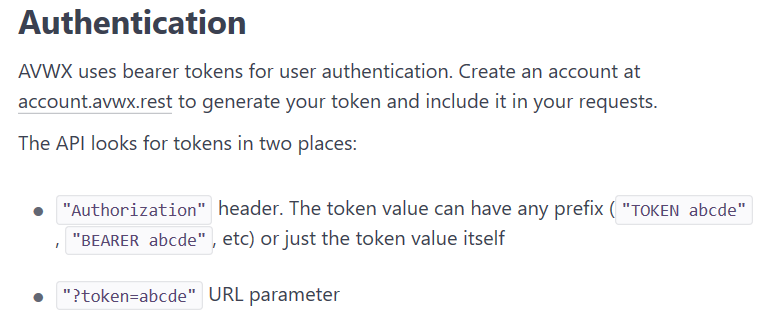


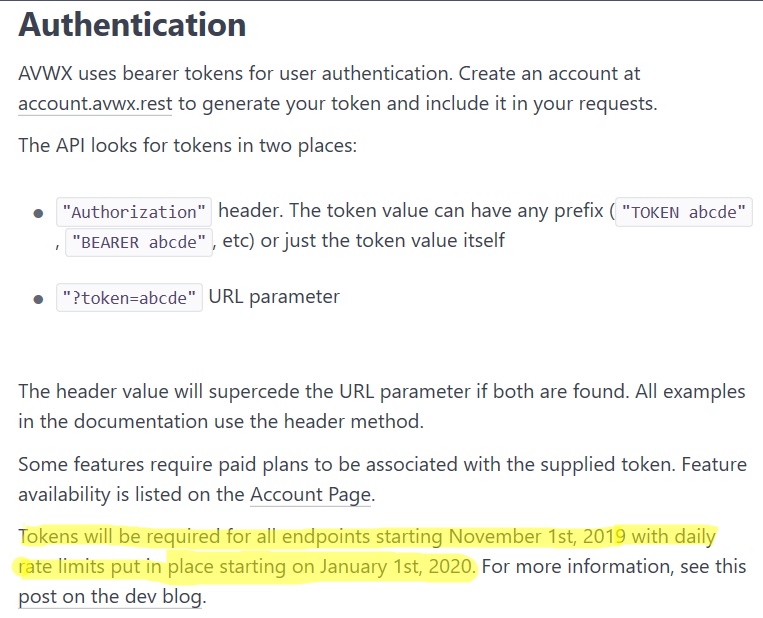
 So, it seems that there is an error. We can now try to see what the error is. After googling how to print the error, there was a stack overflow answer which says to add the on\_failure tag and make this and the on\_error print the error by changing it to:

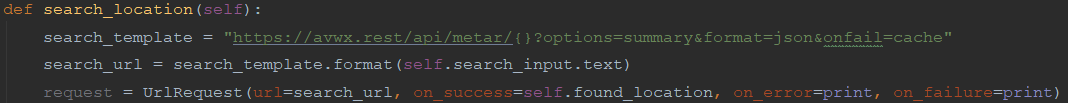
At last, we got an error to deal with at least.

Authorisation error’s usually come in API requests where you need an API key. When I checked the website for the pseudocode and research, it said that no API key was needed. Perhaps, it has been updated.

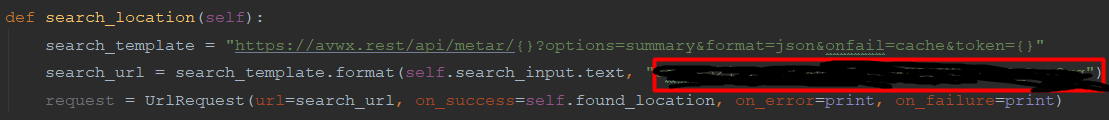
After looking at the docs, it seems like the API now needs authentication via an API token (aka API key)



From further investigation it seems like on November 1st, the system was implemented. I remember writing my pseudocode and researching the API before this and today it is after November 1st.

After creating an account and getting my key, I added the token header in the request template and put my API key in. So, it goes from:

To:



With the API key put inside the red rectangle (I’m hiding it for security reasons of course).

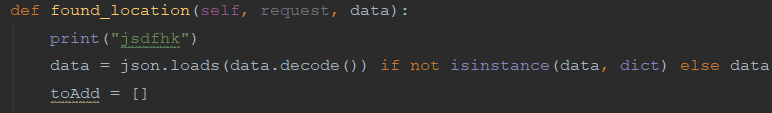
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | What are we testing for | Expected result | Test data | Results | Notes |
| 25 | Check if we can access the API successfully. | We should get a message on the console. | EGLL as the ICAO | Success    We got our message in the console. |  |

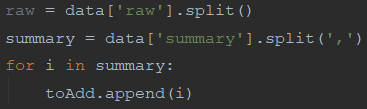
Time to test it again:

That ended up taking the rest of day 8.

### Day 9 – Finishing off the backend

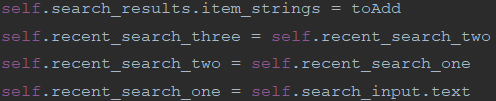
To start the day, I decoded the data using json.loads and prepared the list which will be displayed in the listView



Here I create a list for the raw data containing the raw data split by comma.

The same is applied to the summary of the METAR. Next, each part of the summary is added to the toAdd list.

Finally, we update the list view with the toAdd list so the METAR summary will be displayed to the user.



I have not decided to put in a system to update the user’s details with the new recent search in this version as version 1 is for the fundamental parts of the app and I don’t believe that said system is fundamental to the app.

We move onto our final procedure to fill the recent search dropdown with data. The first line of the procedure may look complicated (because it is). As the user’s data was a string and json.loads() is quite buggy with this sort of stuff, I have decided that doing it manually would be more reliable for now (hence the painful formatting). It basically gets the recent searches from the user details and puts into a list.

Lastly, I update the recent search object properties with the contents of rs to make sure that the recent searches are correct and are visible and functional to the user.

Time to test it:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | What are we testing for | Expected result | Test data | Results | Notes |
| 26 | Check if the METAR is displayed properly on the screen | We should get the details stacked vertically so each part is under each other | EGLL as the ICAO | Success  This is the format which I expected. |  |
| 27 | Checking if the recent searches dropdown is filled correctly | We should get EGCC, VAAH and KJFK respectively | We are using the account with the username ‘h’ | Success |  |
| 28 | Checking if the button to the ICAOFinder works | We should get an error as we have not created that class yet | n/a | Success  We got the desired error |  |

Time to check with our criteria.

### Checking with our criteria

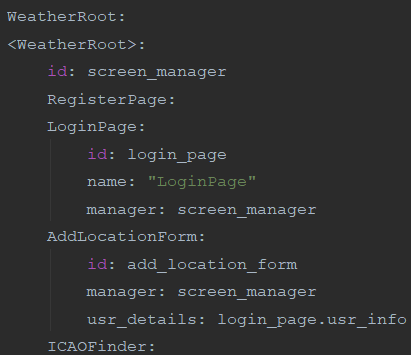
|  |  |  |  |
| --- | --- | --- | --- |
| Criteria | How to check | Completed (Y/N) | Notes |
| METAR search system | A system which presents a decoded accurate METAR in a clear and concise format | Y |  |
| A recent search system | Check if the recent searches have merged from the user’s data and when you click it, it automatically goes in the search box. | Y/N | We have got the fundamentals working but the rest will be in another version as explained. |
| Backend code which searches for the METAR using an API and returns the results | Check-in console with a print statement. It should also show the results on the screen. | Y |  |
| The results to be presented in a clear format | If when we show it to our shareholders, they find it sufficient. | N | Again, I have displayed the fundamentals and the formatting will be done in another version |
| A way for the user to search for the ICAO code | Check that when a button linking the 2 screens is clicked, the screen changes. | Y/N | There is a button which does this, but we need the create the next class for it to work |

On to 1.5!

## Day 10 and 11 – ICAOFinder Class

### Day 10 – The Frontend

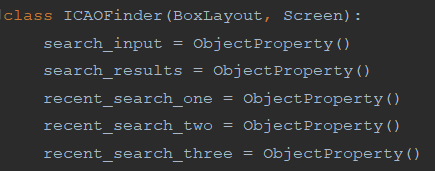
This class will act quite similarly to 1.4. Except from the recent searches and the data which we’ll be using.

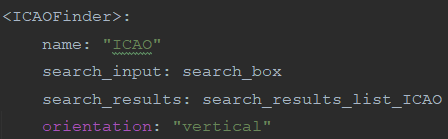


Firstly, we update the root in the kv code so we can navigate to our new class.

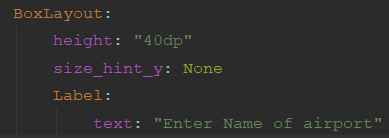
As we’ll not be needing the user details here (for now), there is little point in adding the rest of the details like we did with 1.4.

We also initialise the class in the python code.





Next, we create the class and set its name, the ObjectProperties which we’ll be manipulating in the backend and setting the classes orientation to vertical so the widgets stack properly.

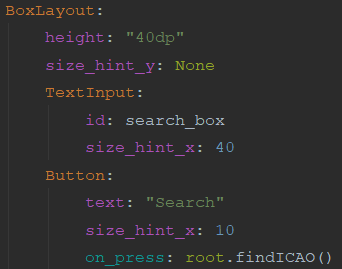


Just like 1.4, we create a title at the top of the screen, this time asking the user for the name of the airport. The height remains the same and the size\_hint\_x is used to make sure that the height can be set.

Now, we should test if this is working.

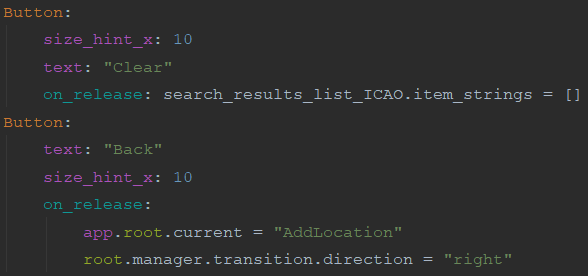
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | What are we testing for | Expected result | Test data | Results | Notes |
| 29 | Check if the class loads and to check if the title text is displayed to the user | The title text will be displayed to the user. If this is displayed then clearly the class has loaded | N/A | Success |  |

Now that we know it works, we can now finish off the rest of the backend.



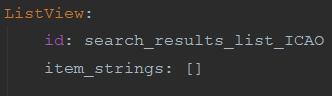
Here, I have created the BoxLayout which will house the rest of the widgets (apart from the ListView). Its height has been set and the input box and the search button has been created.

Both have been id’d so we can call it in the backend and the button has been programmed to run one of the backend procedures once clicked.

Next, we finish off the boxlayout by creating 2 final buttons. One which will clear the contents of the ListView and another which moves the user back to the AddLocation Class when clicked.

There is no need to fiddle with the size hint’s here as if they don’t add up to 100, Kivy will automatically make it do so.

Finally, I create the ListView. This is outside of the boxlayout as if it were inside, it would be next to the back button (as that boxlayout is horizontally orientated) and not underneath (as the widgets in the class are aligned vertically. The horizontal alignment inside the boxlayout overrides the parent inside the boxlayout but outside it remains the same.



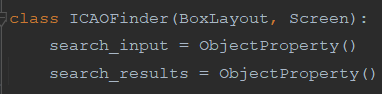
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | What are we testing for | Expected result | Test data | Results | Notes |
| 30 | Check if the widgets have loaded in properly and are displayed correctly to the user. | The title text will be displayed at the top. With the boxlayout underneath (with its widgets aligned horizontally) and the ListView underneath this. | N/A | Success |  |
| 31 | If the Back button sends you back to 1.4 | The screen should be changed to 1.4 | N/A | Success |  |

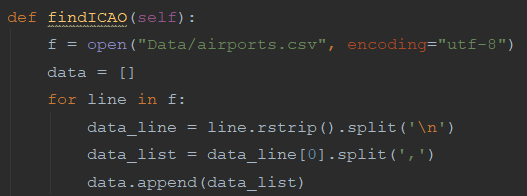
Now we can test the frontend.

The ListView is there, we won’t be able to see it until we put something in it. A bit like the confirmation text from 1.1 and 1.2.

Now on to the backend

### Day 11 – The Backend

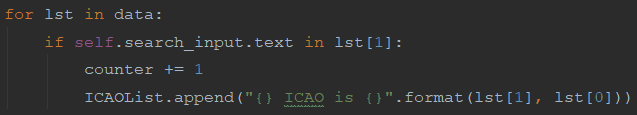
First, we initialise the object properties from the kv code so we can manipulate them here.



Next, we create our main procedure and open our csv file (see note below) so we can search in there (we ensure that its encoding is correct to avoid any errors). Then we initialise a list and iterate over the csv file, putting each line into a list then splitting each line into its individual components (stored as a list) which will be put into our initialised list.

Note – The CSV file has been retrieved from <https://ourairports.com/data/> which has a note to encode its contents in utf-8 for usage. Any other csv files used in this app will also be from this website, unless specified otherwise.

We prepare the start of our list which will be displayed to the user (via the ListView) and a counter which will be used to specify the total number of results.

After that, we create the loop. This iterates over the list made earlier and checks if our search query is part of the airport name (stored in index 1 of said list). If it is then the counter is incremented and these details are added to the ICAOList formatted so it is clear to the user.

Finally, we add the total number of results to the start of the list and update the ListView with the list so the results can be displayed to the user.



Time to test it out.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | What are we testing for | Expected result | Test data | Results | Notes |
| 32 | Check if the search system works | 7 results should appear with Manchester in the name. | Manchester in the search box. | Success |  |
| 33 | Check if the clear button works | The ListView section should be empty | N/A | Success |  |