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

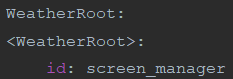
- Any changes needed

- Final Report

# Version 1.0

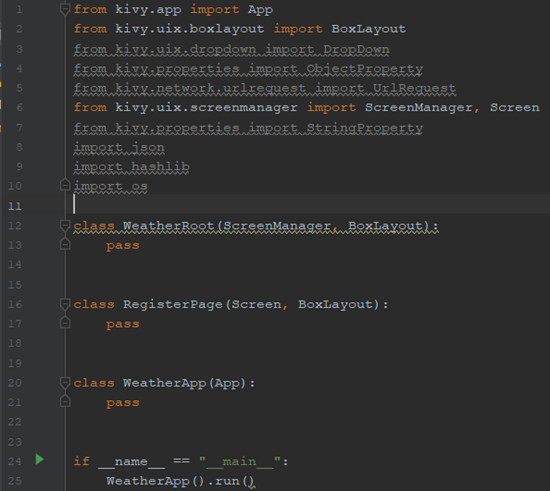
For version 1.0, everything up to actual programming is already in the report.

### 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.

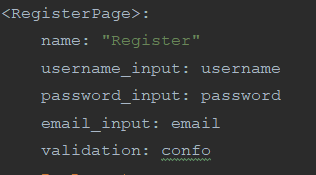
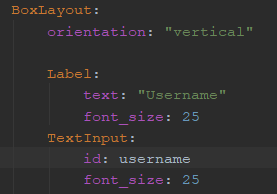
I have also set up the screen manager in the Kivy code for future use.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 next part |

Vcvxcvxcv

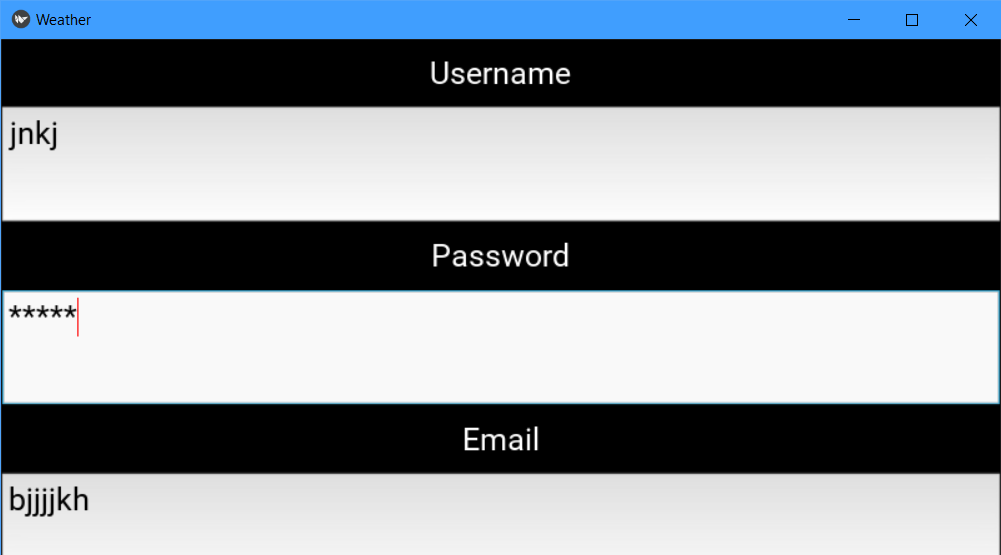
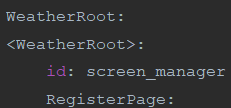
## Day 1, 2 and 3 – RegisterPage Class

### 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.

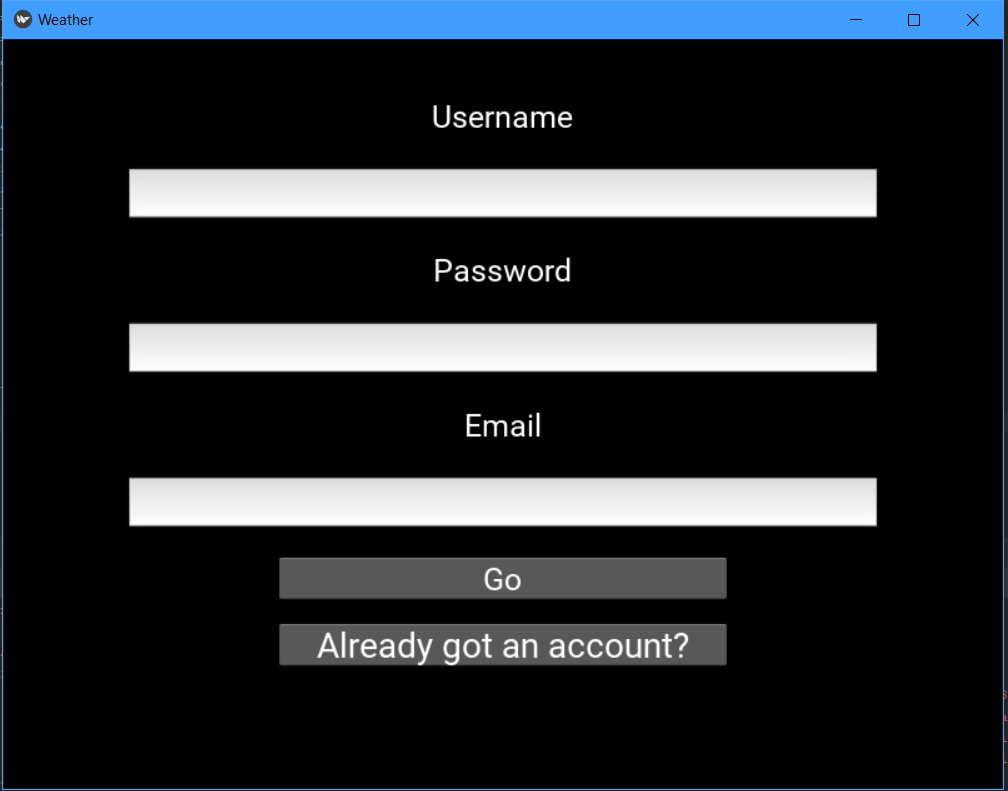
I have also prepared the object properties and the name for switching screens.

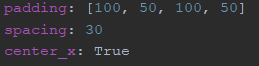


We ensure that this class is a child of the root/screen manager. This is so we can move between screens/classes in our app.

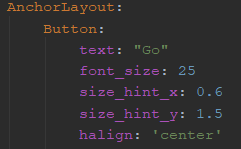
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | Check usage of the text boxes and to make sure nothing is missing | When input text into the boxes, it should be presented clearly and be correctly sized. | Just some random strings | Success (see screenshot) | none |
| 3 | To see if the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

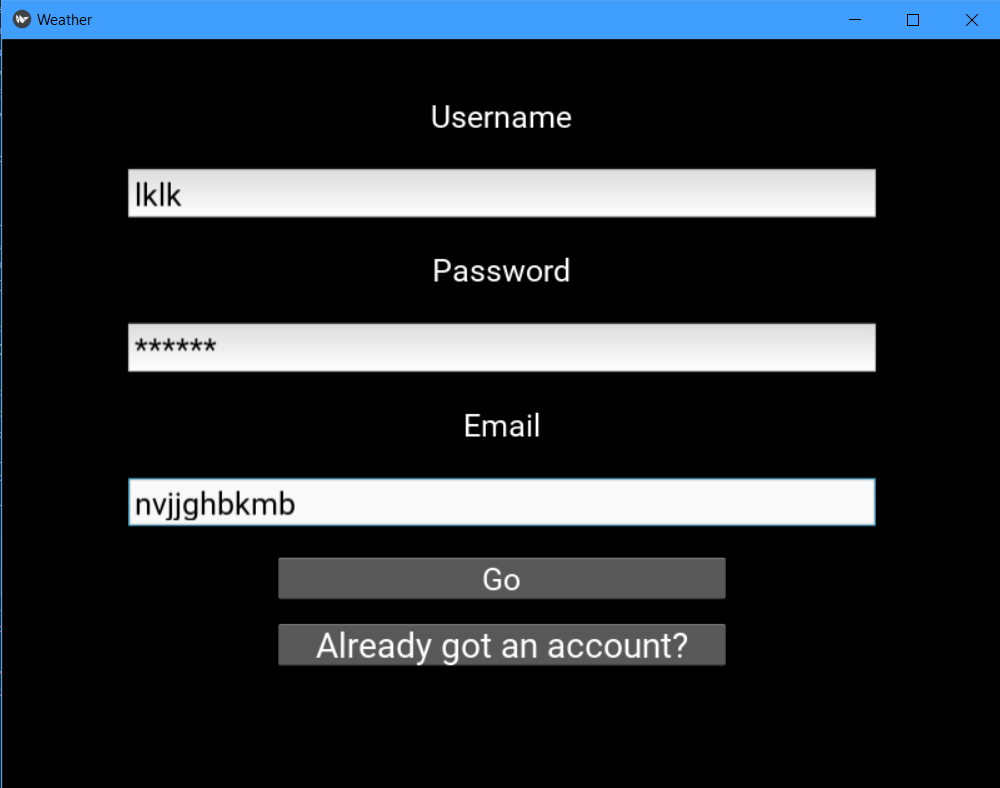
Next, I added padding and spacing to make sure the page looks presentable.





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.

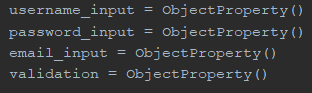




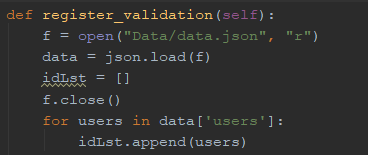
As you can see, the password input box is formatting the input with asterisks and the other inputs are clearly readable.

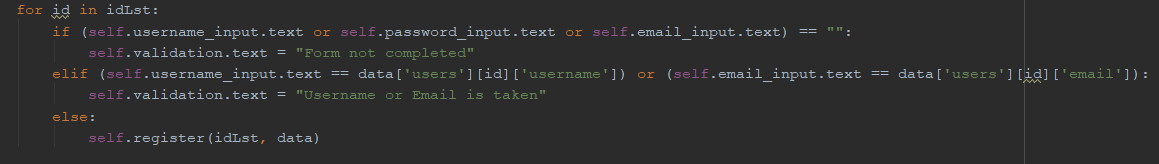
### Day 2 – making the backend.

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 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.

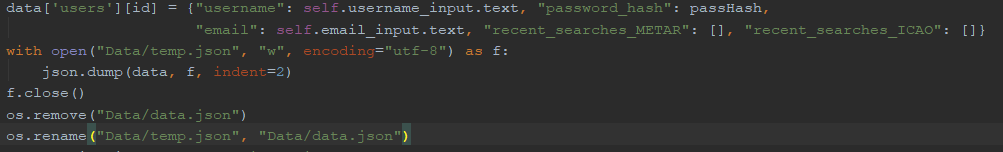


Here we validate that the user has completed the form as well as iterating over the data to check if the user already exists in the json file. If they are then the confirmation text should change based on this.

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.

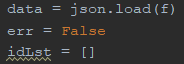
Finally, to conclude the day, I decided to run some tests. The results can be seen below.

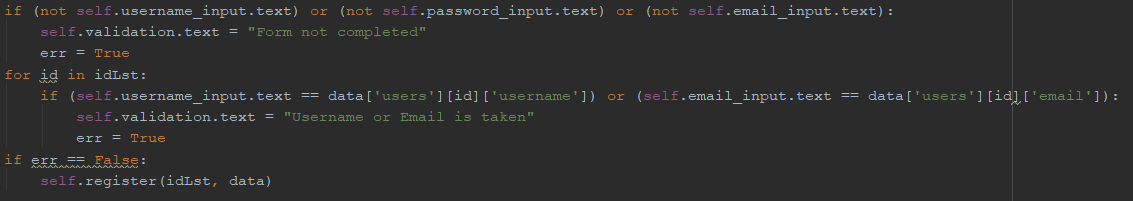
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | What are we testing for | Expected result | Test data | Results | Notes |
| 4 | Test validation process, no username | Expect to see “Form not completed” show up in the confirmation text area and for no registration to continue. | Random strings entered for password and email, nothing in username input box. | Failure, even though it says form not complete:    The registration process still continues. | 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 password input box. | ^ | ^ |
| 6 | Test validation process, no email | ^ | Random strings entered for password and username, noting in 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 username input box. | Success |  |
| 11 | Test validation, no password | ^ | Random strings entered for username and email, nothing in 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 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

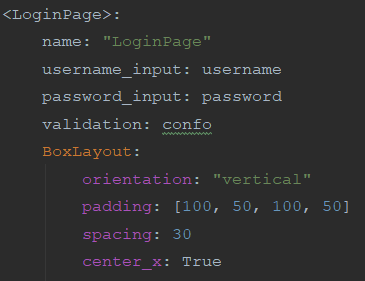
|  |  |  |  |
| --- | --- | --- | --- |
| Criteria | How to check | Completed with tests successful (Y/N) | Notes |
| A register system | We are hoping for a working register system which can add new users and give error messages based on inputs. | Y |  |
| Appropriate validation of the inputs | Test it with all sorts of inputs e.g. missing inputs, existing users etc. | Y |  |
| The data to be kept secure eg: 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 which should return a error message and check if the error message has appeared. | Y |  |
| Success message | ^ but with data which would return a success message. | Y |  |

Now onto the next class.

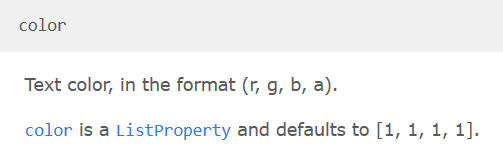
## Day 4, 5 and 6 LoginPage Class

### 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.

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.

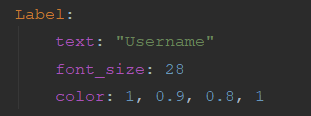
Colour in Kivy, is between 0 and 1 (with allowance for 1 decimal place) for all the rgba values so the default (1,1,1,1) is black with no transparency. As I wanted to add my favourite colour for text on a black background, blanched almond, I had to think of a way to turn the 0 – 255 system into the Kivy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

Blanched almond in the 0-255 system is

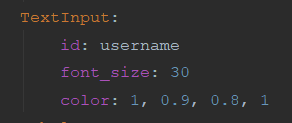
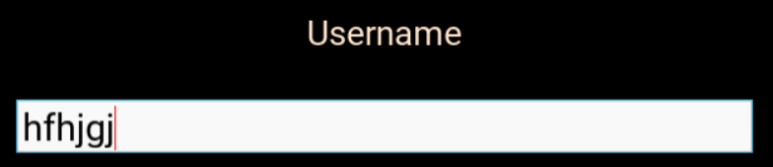
So how about we divide these values by 255 and round to 1 decimal place. Then we get:

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

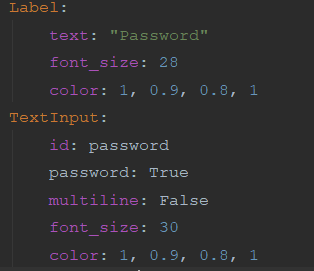
Let’s try it out then:

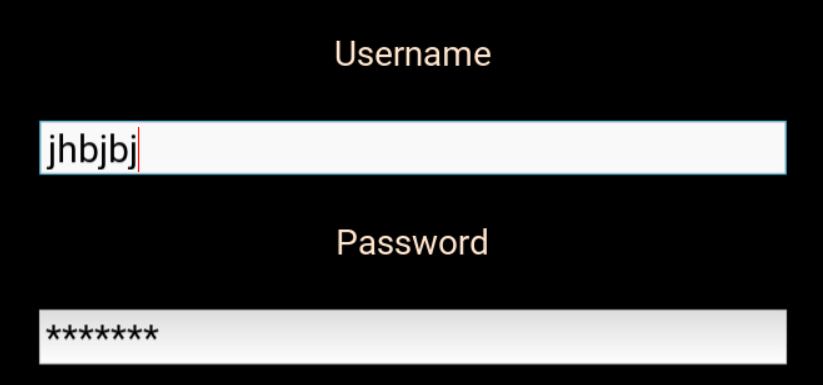


Well that went well, how about we try with a text input

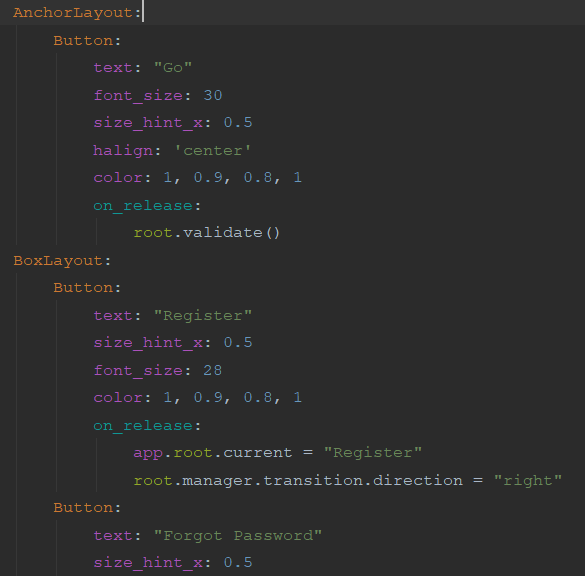


Guess it didn’t work. How about we try colouring a buttons text?

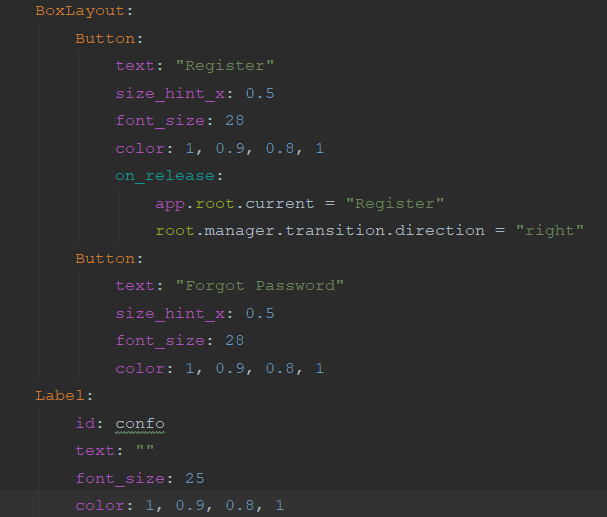
Before we do that, I think we should add our other input and label in – for the password.



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

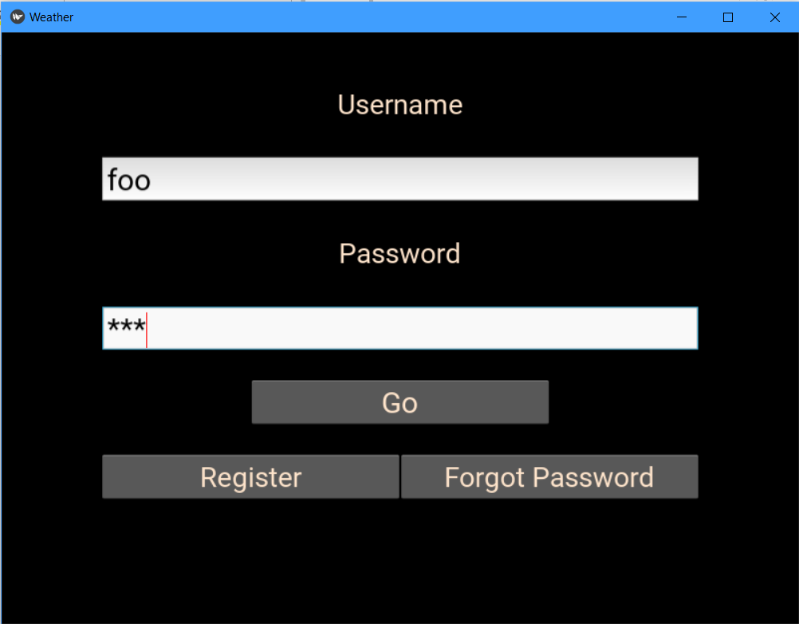


Great. Now we might as well finish off the frontend with the last 2 buttons in a box layout and the confirmation text.

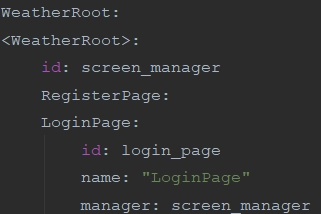


Now that those work, we can run our tests just for procedure’s sake

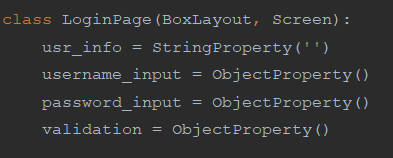
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | What are we testing for | Expected result | Test data | Results | Notes |
| 16 | Check usage of the text boxes and to make sure nothing is missing | When input text into the boxes, it should be presented clearly and be correctly 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 |



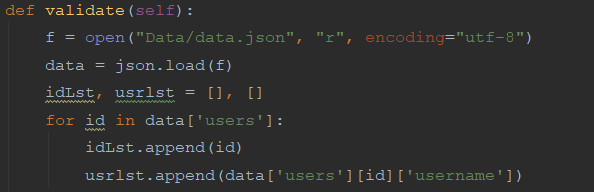
Now that that’s that, let’s move on to the backend of 1.2. I’m sure you realise why blanched almond was a great choice now.

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

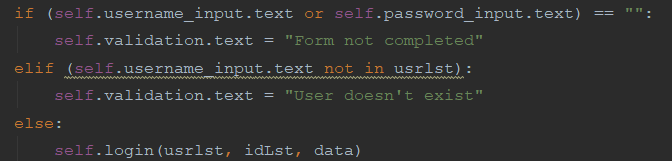
## Day 5 and 6 – The Backend

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 users 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 a 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.

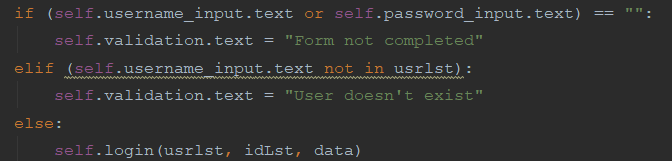
Testing:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 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.

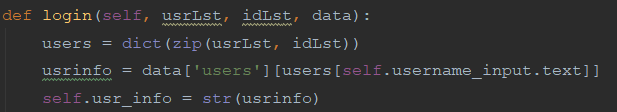
After a quick consolidation session with Quackius V of Mancunium (my rubber duck) we found out that I was dumb and it was quite an easy fix. We replace this:



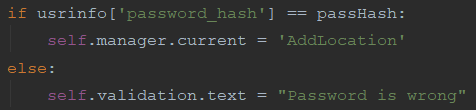
With this:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 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 as random string as 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 a error message then we’re cool. | Y |  |
| On success, go to 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 Backend

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.