CS Dev Log

# Introduction

I will be using an agile development method based on versioning of 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

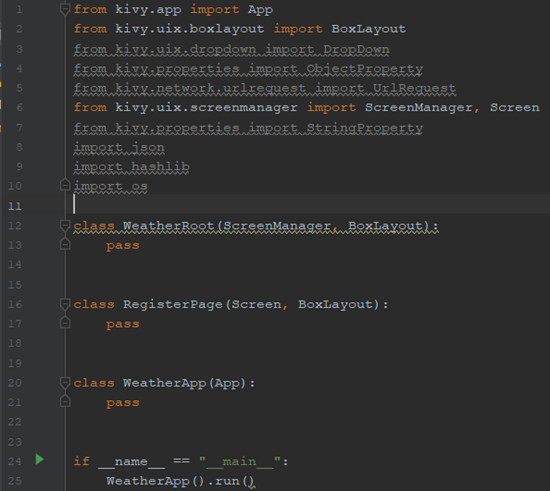
- Final Report

# 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.



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.

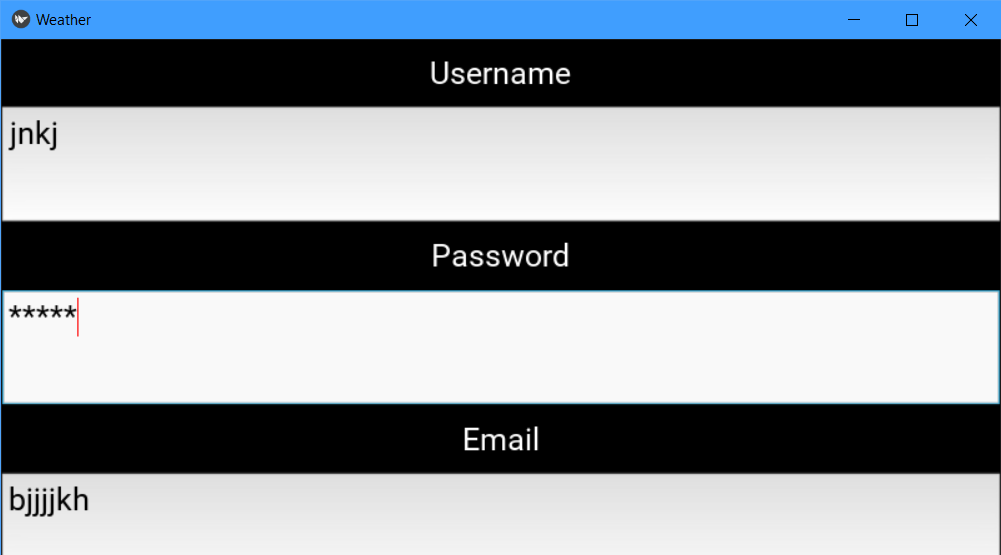
I have also set up the screen manager in the Kivy code for future use.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 |

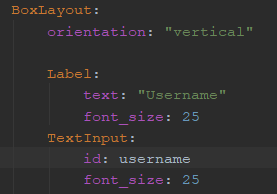
## Day 1, 2 and 3 – RegisterPage Class

### 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.

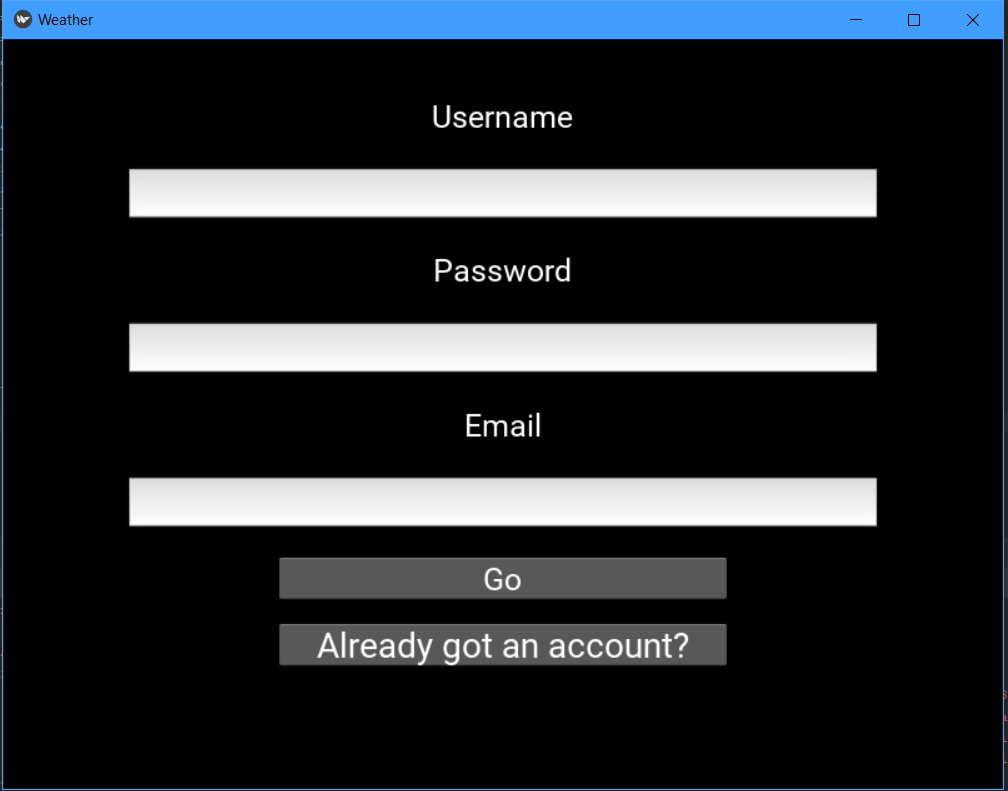


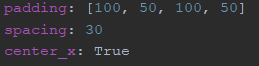
I have created a vertically aligned box layout with 3 labels and 3 input boxes. The password input box has the password formatting added in. I have chosen a font size of 25 for ow to make the text clear and readable.



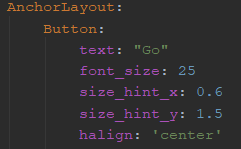
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | Check usage of the text boxes and to make sure nothing is missing | When input text into the boxes, it should be presented clearly and be correctly sized. | Just some random strings | Success (see screenshot) | none |
| 3 | To see if the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

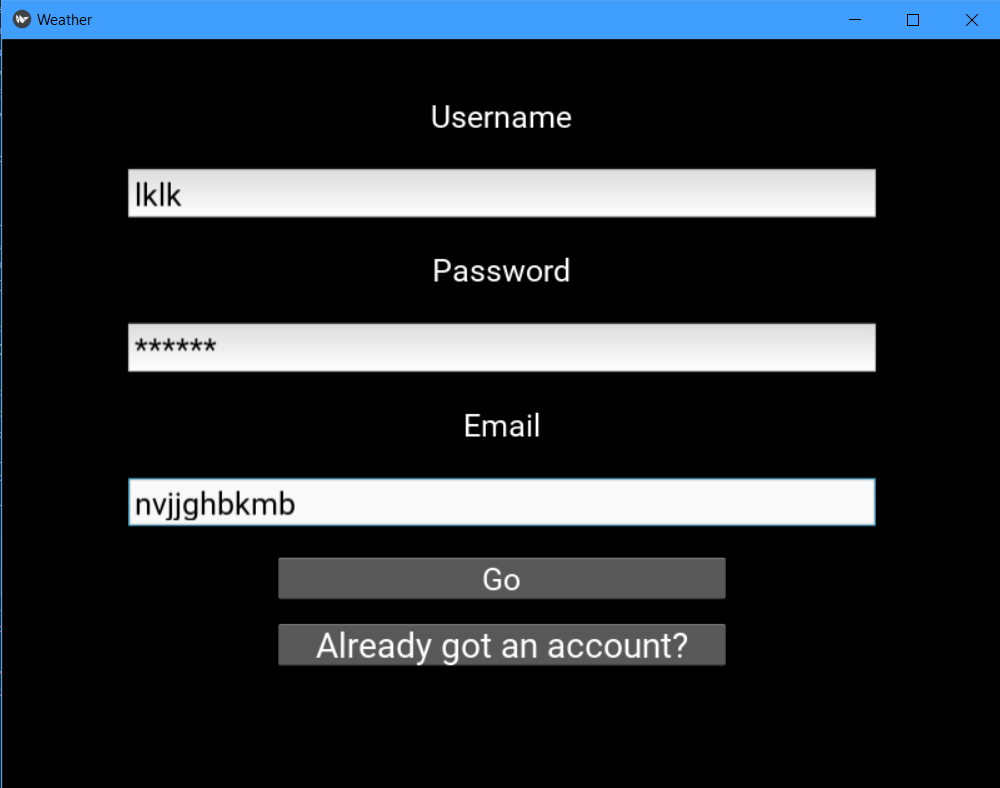
Next, I added padding and spacing to make sure the page looks presentable.





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.

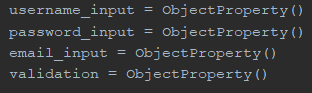




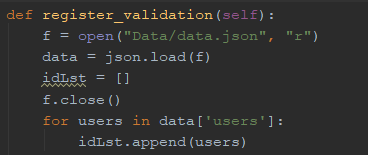
As you can see, the password input box is formatting the input with asterisks and the other inputs are clearly readable.

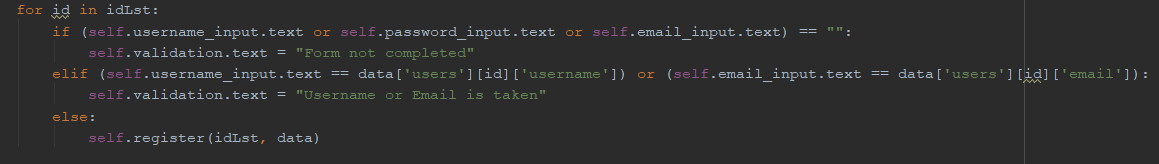
### Day 2 – making the backend.

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.

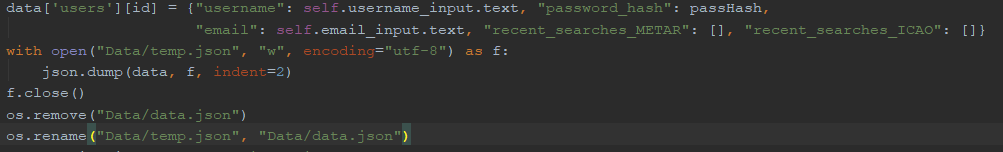


Here we validate that the user has completed the form as well as iterating over the data to check if the user already exists in the json file. If they are then the confirmation text should change based on this.

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.

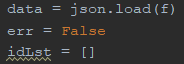
Finally, to conclude the day, I decided to run some tests. The results can be seen below.

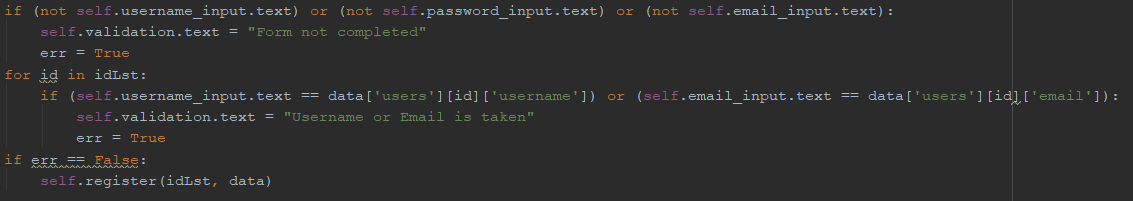
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | What are we testing for | Expected result | Test data | Results | Notes |
| 4 | Test validation process, no username | Expect to see “Form not completed” show up in the confirmation text area and for no registration to continue. | 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 |  |

Now that 1.1 works, we can now move on to 1.2!

## Day 4, 5 and 6 LoginPage Class

### Day 4, Making the frontend