

You have **1** free member-only story left this month. [Sign up for Medium and get an extra one](#)


Building Android Apps With Python: Part -3 (Conclusion)

Step by Step Guide to Build Android Apps using Python


 [Kaustubh Gupta](#) Aug 19, 2020 · 6 min read ★

In the last two parts, we have seen how Kivy and Kivymd make it super easy to develop apps using Python with its drawbacks. We have covered the basics of app development, how to display text, take input, and use buttons to make our app interactive. We have also seen various UI/UIX elements that are supported by Kivy and how they can be easily implemented using Kivy String Builders which are written in a hierarchical format and don't require any explicit import statements. If you haven't read the previous parts, I recommend you have a look at them for better understanding.

Building Android Apps With Python: Part -2
Step by Step Guide to Build Android Apps using Python
medium.com



Building Android Apps With Python: Part -1
Step by Step Guide to Build Android Apps using Python
towardsdatascience.com



In this part, we will cover all the remaining frequently used elements in Kivymd, and in another part, we will build our capstone app which will fetch weather-related information using weather API, convert that app into Android APK and deploy it on an online platform. Isn't this cool?

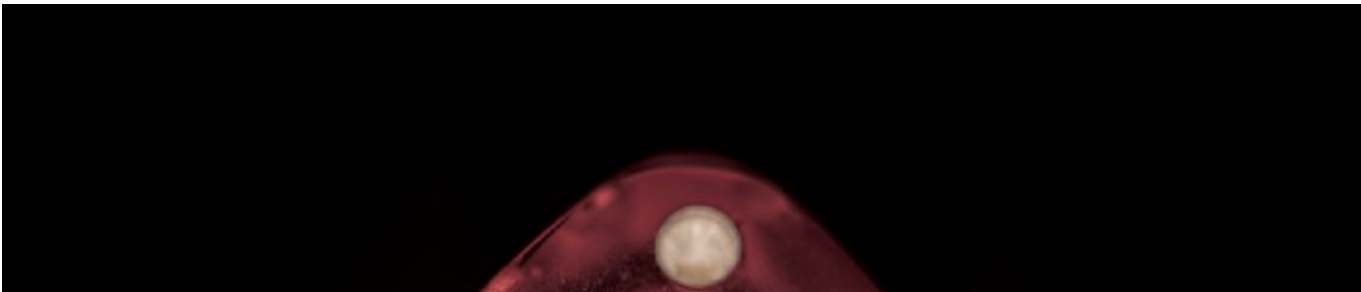




Photo by [Mike Szczepanski](#) on [Unsplash](#)

MDDialog

Last time we built a basic app that takes a password as input, compares it with our keyword “root” and displays success or failure message but as text on the screen. What if a dialog box pops up which not only displays the result of the action but gives more options if applicable? This type of function can be made with the help of MDDialog. Let’s look at its implementation:

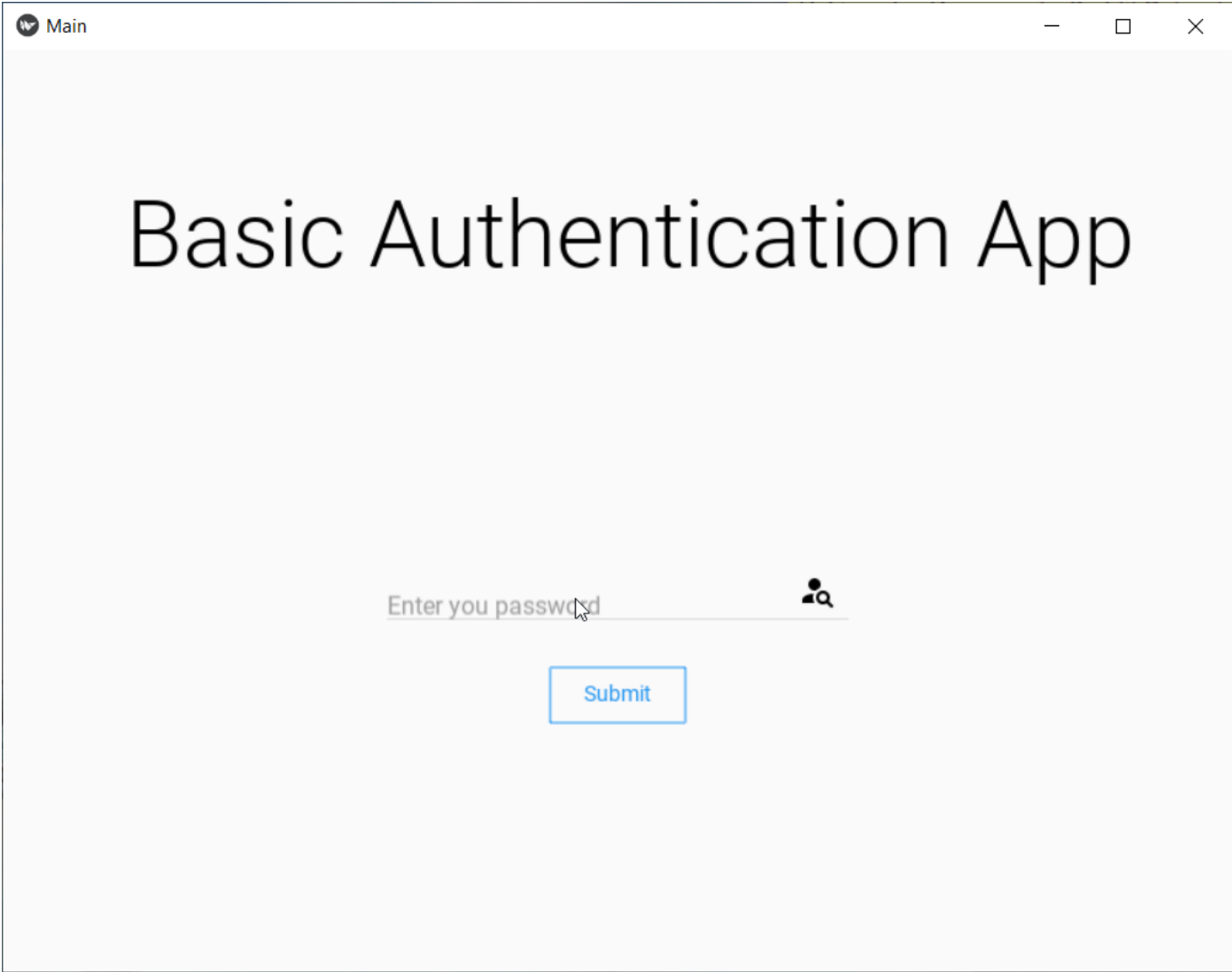
```
1  from kivymd.app import MDApp
2  from kivy.lang import Builder
3  from kivy.properties import ObjectProperty
4  from kivymd.uix.dialog import MDDialog
5  from kivymd.uix.button import MDFlatButton
6
7  kv = """
8  Screen:
9      in_class: text
10     MDLabel:
11         text: 'Basic Authentication App'
12         font_style: 'H2'
13         pos_hint: {'center_x': 0.6, 'center_y': 0.8}
14     MDTextField:
15         id: text
16         hint_text: 'Enter you password'
17         helper_text: 'Forgot your password?'
18         helper_text_mode: "on_focus"
19         pos_hint: {'center_x': 0.5, 'center_y': 0.4}
20         size_hint_x: None
21         width: 300
22         icon_right: "account-search"
23         required: True
24
25     MDRectangleFlatButton:
26         text: 'Submit'
27         pos_hint: {'center_x': 0.5, 'center_y': 0.3}
28         on_press:
29             app.auth()
30
31     MDLabel:
32         text: ''
33         id: show
34         pos_hint: {'center_x': 1.0, 'center_y': 0.2}
35
36  """
37
38
39  class Main(MDApp):
40      in_class = ObjectProperty(None)
41
```

```
42     def build(self):
43         return Builder.load_string(kv)
44
45     def auth(self):
46         if self.root.in_class.text == 'root':
47             # label = self.root.ids.show
48             # label.text = "Sucess"
49             self.dialog = MDDialog(title='Password check',
50                                     text="Sucess !", size_hint=(0.8, 1),
51                                     buttons=[MDFlatButton(text='Close', on_release=self.c
52                                                 MDFFlatButton(text='More'))
53                                     )
54             self.dialog.open()
55         else:
56             # label = self.root.ids.show
57             # label.text = "Fail"
58             self.dialog.text = 'Fail !'
59             self.dialog.open()
60
61     def close_dialog(self, obj):
62         self.dialog.dismiss()
63
64
65 Main().run()
```

Auth_with_dialog.py hosted with ❤ by GitHub [view raw](#)

See how the Dialog was called when the comparison was made

The output looks like this:



GIF by Author

Various properties can be modified in a dialog box like:

1. various types of events including on_pre_open, on_open, on_pre_dismiss, on_dismiss
2. “title” of the dialog box. In the example, the Password check is the title of the box.

3. “text” of the box. Here we have success or failure as the text.
4. control the “radius” of the box and make it round, elliptical.
5. add buttons, list, make it a form that can take inputs, and many more.

Read more about these features [here](#).

MDList (One, Two & Three line list item)

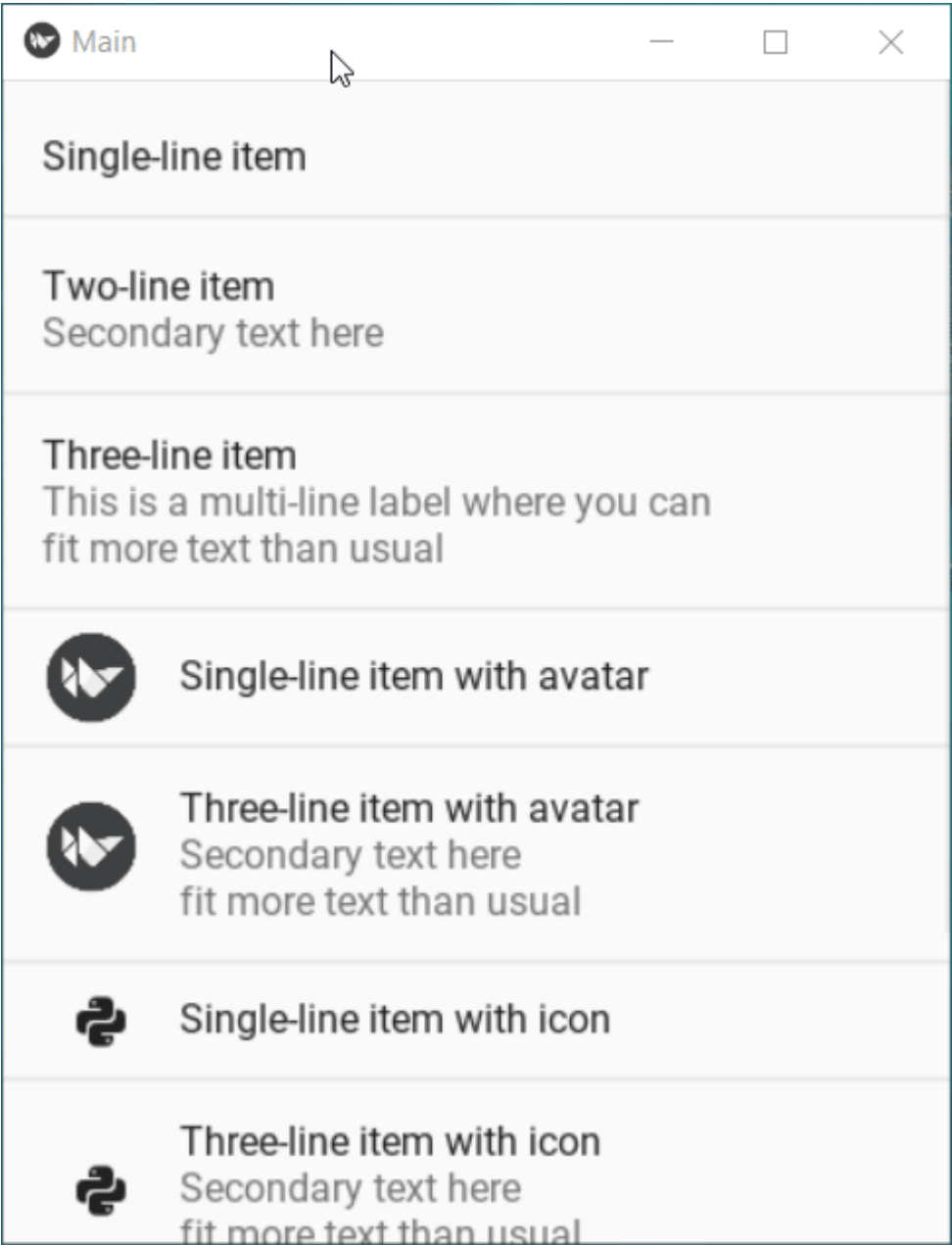
The list is one of the most commonly used entities in app development. It is a vertical continuous index of text with images. MDList is combined with a scroll view and list items to make a functional list. We have different types of list items in Kivymd ranging from OneLineListItem to ThreeLineAvatarIconListItem. The example below covers all type of items in a list:

```
1  from kivymd.app import MDApp
2  from kivy.lang import Builder
3
4  kv = """
5  Screen:
6      ScrollView:
7          MDList:
8              OneLineListItem:
9                  text: "Single-line item"
10             TwoLineListItem:
11                 text: "Two-line item"
12                 secondary_text: "Secondary text here"
13             ThreeLineListItem:
14                 text: "Three-line item"
15                 secondary_text: "This is a multi-line label where you can"
16                 tertiary_text: "fit more text than usual"
17             OneLineAvatarListItem:
18                 text: "Single-line item with avatar"
19                 ImageLeftWidget:
20                     source: "data/logo/kivy-icon-256.png"
21             ThreeLineAvatarListItem:
22                 text: "Three-line item with avatar"
23                 secondary_text: "Secondary text here"
24                 tertiary_text: "fit more text than usual"
25                 ImageLeftWidget:
26                     source: "data/logo/kivy-icon-256.png"
27             OneLineIconListItem:
28                 text: "Single-line item with icon"
29                 IconLeftWidget:
30                     icon: "language-python"
31             ThreeLineIconListItem:
32                 text: "Three-line item with icon"
33                 secondary_text: "Secondary text here"
34                 tertiary_text: "fit more text than usual"
35                 IconLeftWidget:
36                     icon: "language-python"
37             OneLineAvatarIconListItem:
38                 text: "One-line item with avatar or icon"
39                 IconLeftWidget:
40                     icon: "plus"
41                 IconRightWidget:
42                     icon: "minus"
43             ThreeLineAvatarIconListItem:
44                 text: "Three-line item with avatar or icon"
45                 secondary_text: "Secondary text here"
46                 tertiary_text: "fit more text than usual"
47                 IconLeftWidget:
```

```
48         icon: "plus"
49     IconRightWidget:
50         icon: "minus"
51     """
52
53
54 class Main(MDApp):
55
56     def build(self):
57         return Builder.load_string(kv)
58
59
60 Main().run()
```

lists.py hosted with ❤ by GitHub view raw

Look at its output:



All different types of list items

The basic difference between avatar and icon is that icons are mostly predefined, freely available and they don’t need to be explicitly referenced via source. Avatar is more of an image that is referenced to a profile though it can be used in a list item. The items contain text which can be modified using the text properties of MDLabel.

MDDataTables

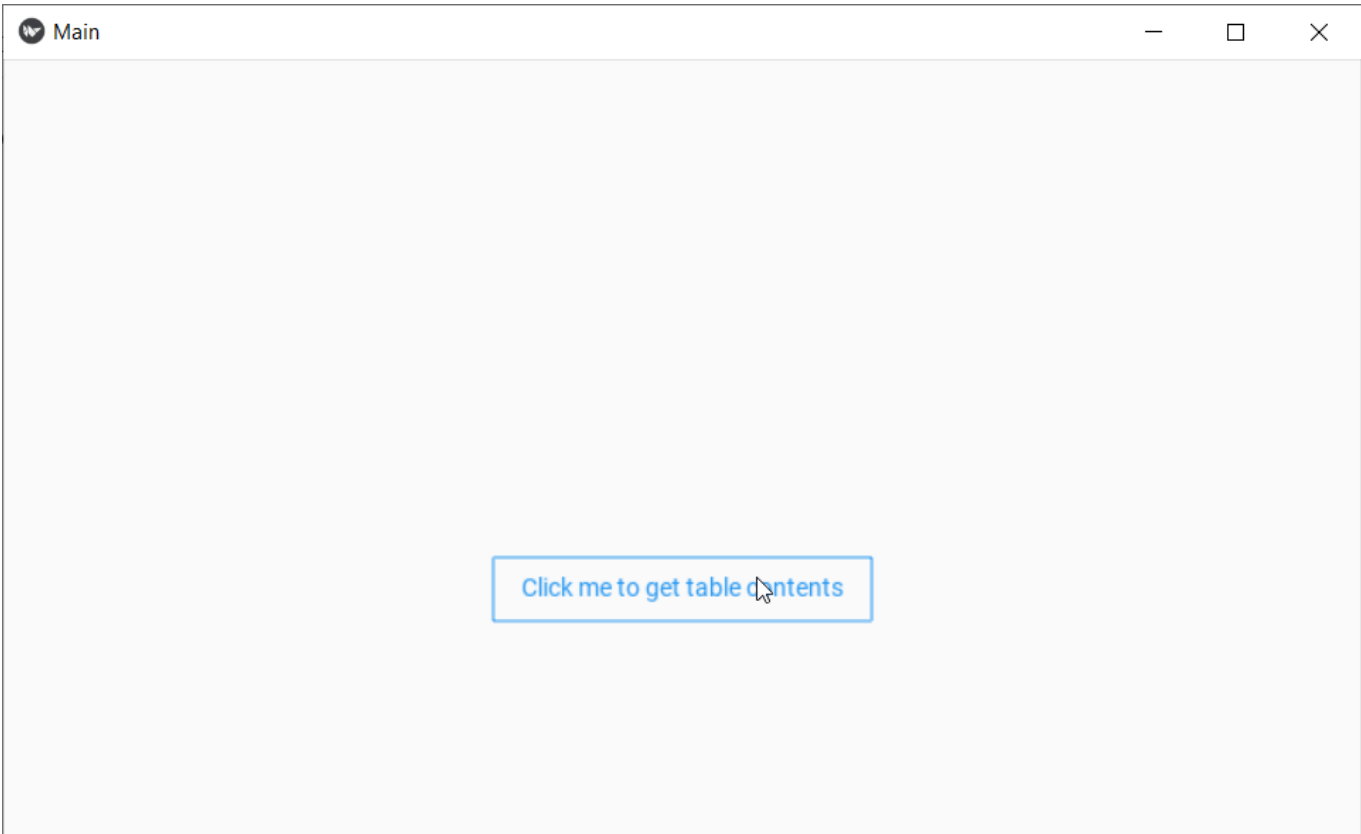
Datatables are used to display tabular data in the form of rows and columns. It is a simple yet powerful utility offered by Kivymd. It can be linked with a button or any other action. The row items can be combined with checkboxes making a to-do list in a table format. You can add

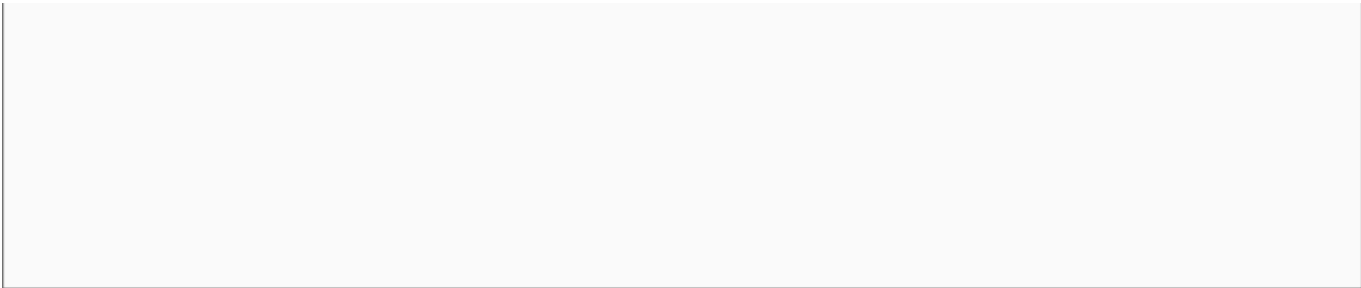
pagination, control the number of rows to be displayed. Let’s check it’s code:

```
1  from kivymd.app import MDApp
2  from kivy.lang import Builder
3  from kivymd.uix.datatables import MDDataTable
4  from kivy.metrics import dp
5
6  kv = """
7  Screen:
8      MDRectangleFlatButton:
9          text: 'Click me to get table contents'
10         pos_hint: {'center_x': 0.5, 'center_y': 0.5}
11         on_press: app.table()
12
13  """
14
15
16  class Main(MDApp):
17
18      def table(self):
19          self.tables = MDDataTable(size_hint=(0.9, 0.6),
20                                   use_pagination=True,
21                                   column_data=[
22                                       ("No.", dp(30)),
23                                       ("Column 1", dp(30)),
24                                       ("Column 2", dp(30)),
25                                       ("Column 3", dp(30)),
26                                       ("Column 4", dp(30)),
27                                       ("Column 5", dp(30)),
28                                   ],
29                                   row_data=[
30                                       (f"{i + 1}", "1", "2", "3", "4", "5") for i in range(10)
31                                   ],
32                                   )
33
34          self.tables.open()
35
36      def build(self):
37          return Builder.load_string(kv)
38
39  Main().run()
```

datatables.py hosted with ❤ by GitHub [view raw](#)

And the interesting output:





GIF by Author

MDToolbar

You all have seen in android apps we have a top stripped portion where the name of the app is mentioned with a stack of three lines on the left and three dots on the right side. It is called a toolbar and it is very essential in an app to make it more easily accessible and organize different actions. As always Kivymd comes with options and this one also has two options: Top toolbar and bottom toolbar. Both have their advantages and features but for this series, we will focus on Top toolbar. If you want to know more about the bottom toolbar then follow [this link](#). A simple toolbar is coded as follows:

```
1 kv = """
2 Screen:
3     BoxLayout:
4         orientation: 'vertical'
5         MDToolbar:
6             title: "MDToolbar"
7             left_action_items: [["menu", lambda x: app.callback()]]
8             right_action_items: [["dots-vertical", lambda x: app.callback()]]
9         Widget:
10
11
12 """
```

toolbar_without_navigation.py hosted with ❤ by GitHub [view raw](#)

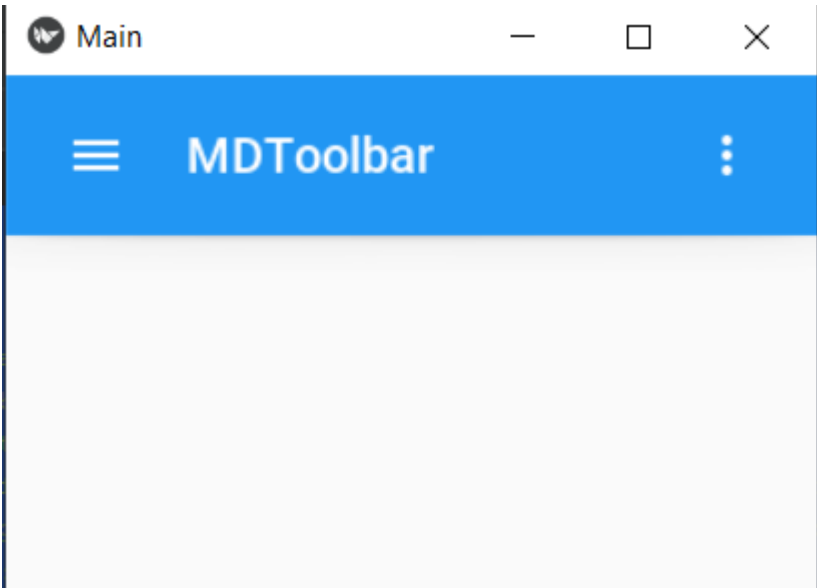


Image by Author

Now if you click on the 3 dots menu, it will crash the app as it requires a callback. Callback defines what action should be taken for that particular event. To make a functional menu, we need something called Navigation Drawer, let’s see what is it.

MDNavigationDrawer

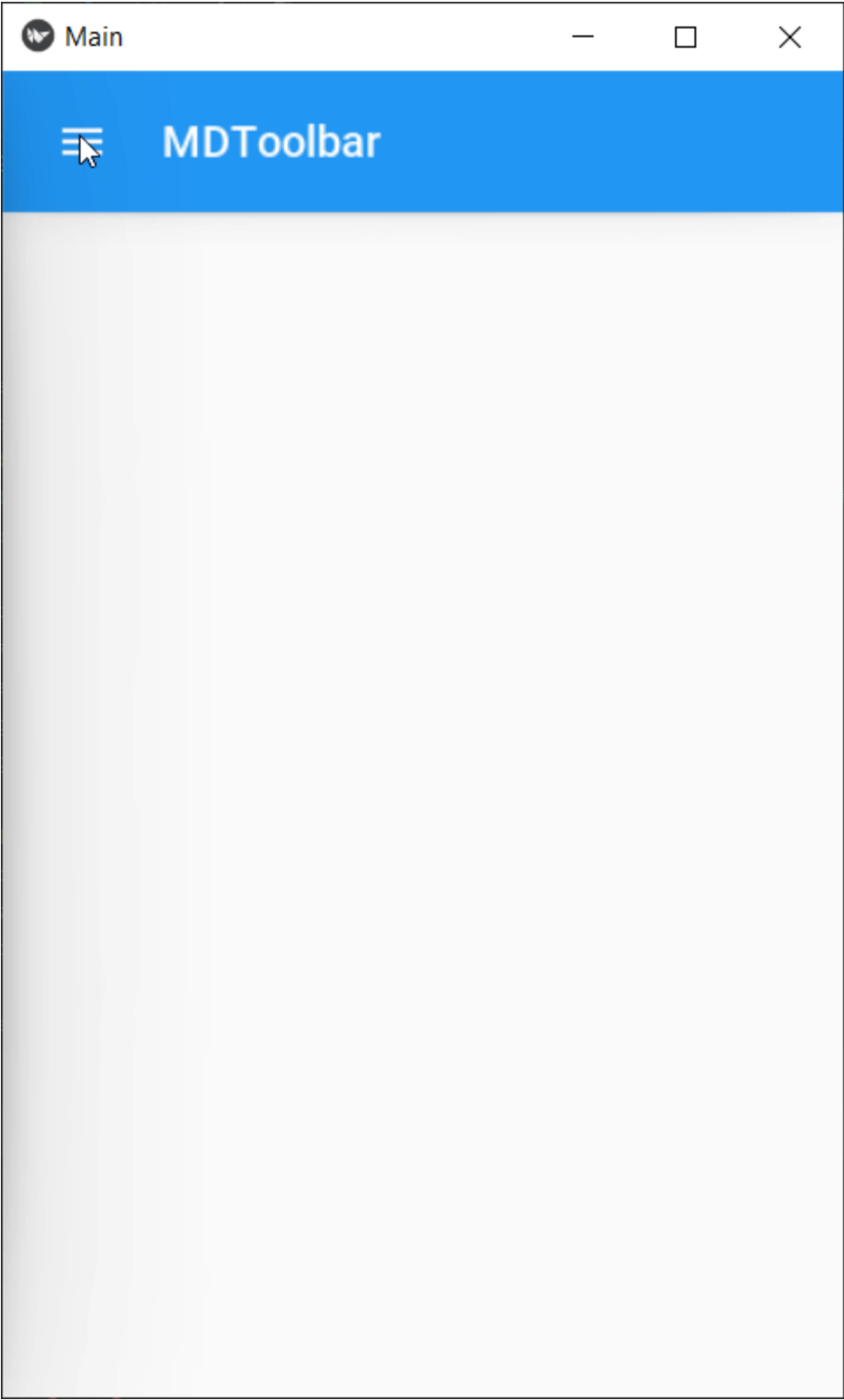
This is where things get interesting. Now we can finally build a menu in our app with different sections to browse. Navigation Drawer makes it possible to access different destinations in our app. Before I show the code explanation, there is one more concept here. There are different types of layout Kivymd offers. **The layout** is a type of blueprint where all the elements can be arranged. The MDToolbar is usually placed in a BoxLayout. Learn more about layouts [here](#). Now look at this code carefully:

```
1  from kivymd.app import MDApp
2  from kivy.lang import Builder
3
4  kv = """
5  Screen:
6      BoxLayout:
7          orientation: 'vertical'
8          MDToolbar:
9              title: "MDToolbar"
10             left_action_items: [["menu", lambda x: nav_draw.set_state()]]
11         Widget:
12
13     MDDrawer:
14         id: nav_draw
15         orientation: "vertical"
16         padding: "8dp"
17         spacing: "8dp"
18
19         AnchorLayout:
20             anchor_x: "left"
21             size_hint_y: None
22             height: avatar.height
23
24             Image:
25                 id: avatar
26                 size_hint: None, None
27                 size: "56dp", "56dp"
28                 source: "data/logo/kivy-icon-256.png"
29
30             MDLabel:
31                 text: "Kaustubh Gupta"
32                 font_style: "Button"
33                 size_hint_y: None
34                 height: self.texture_size[1]
35
36             MDLabel:
37                 text: "youreamil@gmail.com"
38                 font_style: "Caption"
39                 size_hint_y: None
40                 height: self.texture_size[1]
41
42         ScrollView:
43             MDList:
44                 OneLineAvatarListItem:
45                     on_press:
46                         nav_draw.set_state("close")
47
48                     text: "Home"
49                     IconLeftWidget:
50                         icon: "home"
51
52                 OneLineAvatarListItem:
53                     on_press:
54                         nav_draw.set_state("close")
55
56                     text: "About"
57                     IconLeftWidget:
58                         icon: 'information'
```



```
59
60     Widget:
61     """
62
63
64     class Main(MDApp):
65
66         def build(self):
67             return Builder.load_string(kv)
68
69
70     Main().run()
```

toolbar_with_menu.py hosted with ❤ by GitHub [view raw](#)



GIF by Author

Here I have placed the MDToolbar in a BoxLayout while in MDNavigationDrawer, I have used an AnchorLayout so that I can have an introduction section in the menu bar. After this section, we have the MDList having two destinations, Home and About. Now that we have come so far in our android app journey, let’s see how we can work with different screens and make use of the same MDNavigation bar.

Switching Screens (ScreenManager)

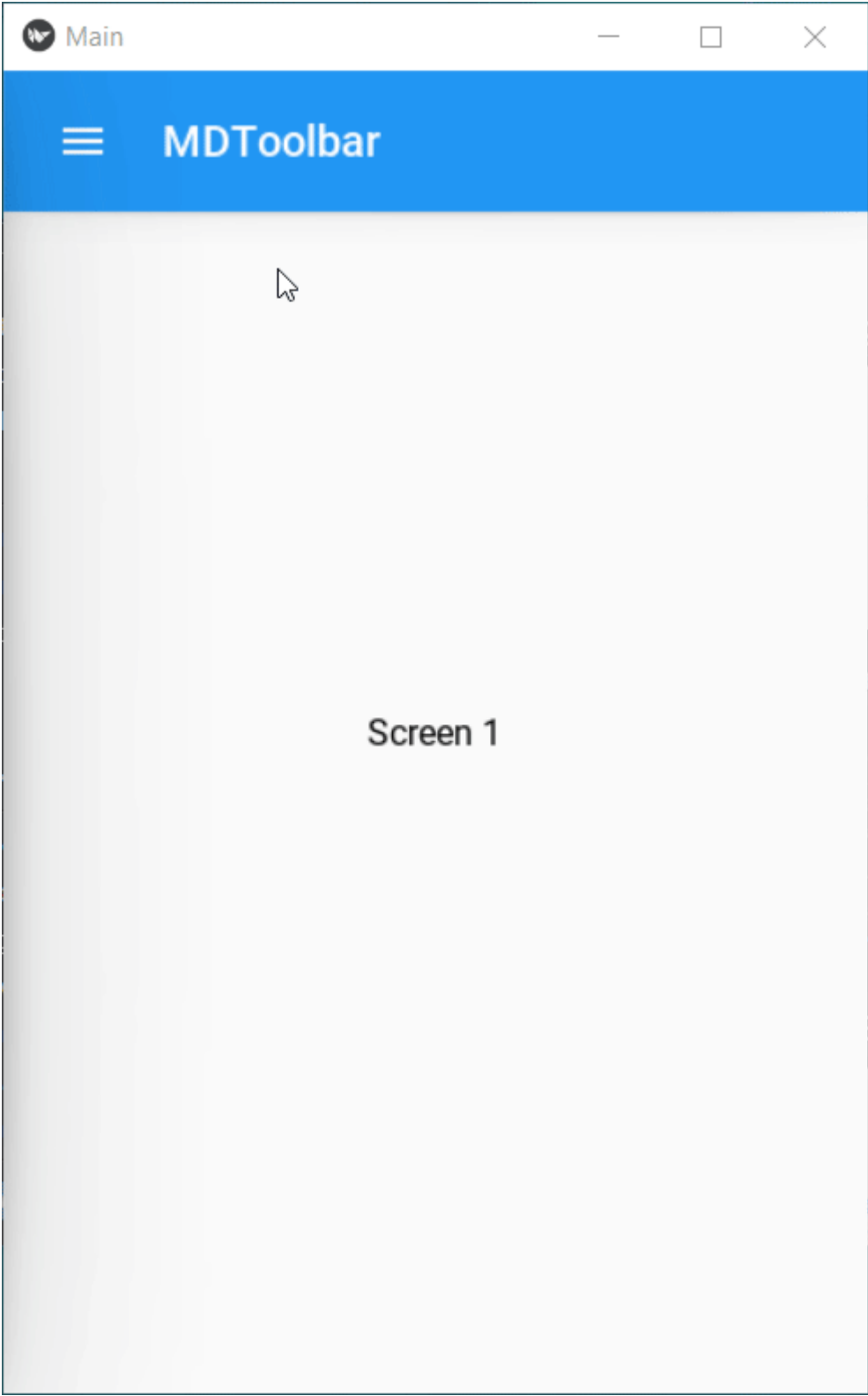
As the name suggests, **ScreenManager** is used to manage all the screens we want to include in our app. It holds all the screens with a unique id and that id can be used to switch screens on the action. To use the same toolbar in all screens, we will place the Screen Manager and **NavigationDrawer** inside the **NavigationLayout**. Now, if we go to the Navigation bar and tap on about or information, nothing happens but we can bind this to switch screen. Just refer to the id of the screen manager and change the current screen to the screen you want. See an example here:

```
1 kv = """
2 Screen:
3     BoxLayout:
4         orientation: 'vertical'
5         MDToolbar:
6             title: "MDToolbar"
7             left_action_items: [["menu", lambda x: nav_draw.set_state()]]
8         Widget:
9     NavigationLayout:
10         ScreenManager:
11             id: screen_manager
12         Screen:
13             name: "scr1"
14             MDLabel:
15                 text: "Screen 1"
16                 halign: "center"
17
18         Screen:
19             name: "scr2"
20             MDLabel:
21                 text: "Screen 2"
22                 halign: "center"
23
24         MDNavigationDrawer:
25             id: nav_draw
26             orientation: "vertical"
27             padding: "8dp"
28             spacing: "8dp"
29
30         AnchorLayout:
31             anchor_x: "left"
32             size_hint_y: None
33             height: avatar.height
34
35             Image:
36                 id: avatar
37                 size_hint: None, None
38                 size: "56dp", "56dp"
39                 source: "data/logo/kivy-icon-256.png"
40
41             MDLabel:
42                 text: "Kaustubh Gupta"
43                 font_style: "Button"
44                 size_hint_y: None
45                 height: self.texture_size[1]
46
47             MDLabel:
48                 text: "youreamil@gmail.com"
49                 font_style: "Caption"
50                 size_hint_y: None
51                 height: self.texture_size[1]
52
53         ScrollView:
54             MDList:
55                 OneLineAvatarListItem:
56                     on_press:
57                         nav_draw.set_state("close")
```

```
58         screen_manager.current = "scr1"
59
60         text: "Home"
61         IconLeftWidget:
62             icon: "home"
63
64         OneLineAvatarListItem:
65             on_press:
66                 nav_draw.set_state("close")
67                 screen_manager.current = "scr2"
68             text: "About"
69             IconLeftWidget:
70                 icon: 'information'
71
72     Widget:
73         """
```

screen_switch.py hosted with ❤ by GitHub

view raw



GIF by Author

Conclusion and What’s Next

In this part of the series, we covered more elements used in app development. We learned how to make the app interactive using dialog boxes, how to create the toolbar, and switch screens on a particular action/event. In an upcoming article, I will make an app, convert it to APK and deploy it on a cloud platform so stay tuned for that article. If you liked

this development series make sure to follow me on medium so you receive notifications about upcoming articles, comment down if you have any queries, and with that said, Sayonara!


[Linkedin](#), [Github](#)

Update: The conversion/deployment article of Kivy apps is live now!

3 Ways to Convert Python App into APK

Concluding Building Android Apps in Python Series!


towardsdatascience.com



Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

 Get this newsletter

You'll need to sign in or create an account to receive this newsletter.

- Python

Android

Kivy

Kivymd

Android App Development