# Lab 14

Instructor: Sidra Khatoon E-mail: skhatoon[@uit.edu](mailto:ad@uit.edu)

# Objective

# The objective of lab is exploring provider and riverpod

# .

**Student Information**

|  |  |
| --- | --- |
| **Student Name** |  |
| **Student ID** |  |
| **Date** |  |

**Assessment**

|  |  |
| --- | --- |
| **Marks Obtained** |  |
| **Remarks** |  |
| **Signature** |  |

# Objective

# The objective of lab is exploring Drawer and tab Navigation.

# Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

# How to Submit?

Submit lab work using Teams.

**Learn State Management in Flutter by Building a Simple Todo App:**

State management is a complex topic in mobile application development. But it's also a necessary topic that plays a major role in building dynamic mobile apps.

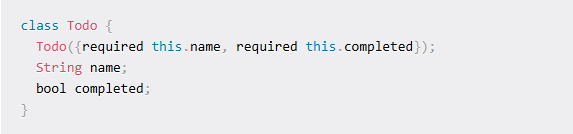
First we create new app with basic counter application. Rename class with Todo list.

Let's build the core functionality of our app.

We need a Todo class. This class will define the properties of a todo. In our case, we'll have the following items:

1. Name of the todo
2. Status of the todo (Completed)

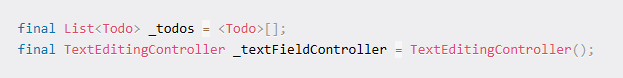
Let's define a Todo class with the above properties:



Add the above code at the bottom of the main.dart file.

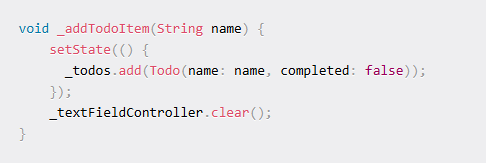
**How to Add a Todo**

In the body of the build method, set the children property to an empty array. Remove the two Text widgets inside that children property. Now we'll replace the counter variable with a todo list.



The first line is the todo list and the second line defines the controller to get the name of the todo from the user.

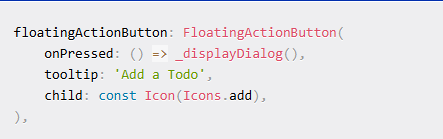
Remove the \_incrementCounter method and add the method to add a todo:



So far we have defined our todo list and an input controller. We've also created a method that accepts input text and adds that to the todo list with a completed status set to false and a clear input field.

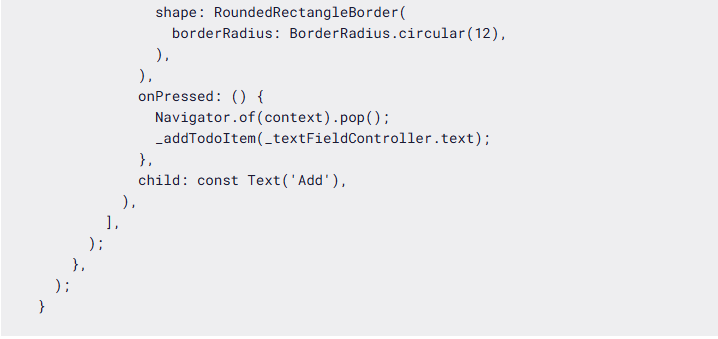
The reason we have used the setState method is to refresh the UI after we update the todo list. As our component is a stateful widget, whenever a change in state is detected, the UI will render again with the updated state.

We have built the functionality code to add a todo. Let's build the UI code. Let's ask the user the name of the todo on pressing the Floating action button at the bottom right. When the user tries to save the todo, we'll call the \_addTodoItem method defined above.



n the above method, we have changed the onPressed property to call the \_displayDialog method. As it's not defined yet, it'll show an error. We'll define the method next. We have also changed the tooltip property to "Add a Todo".

Here's the code (\_displayDialog method) to show a dialog box with an input field, add, and cancel button. Add this method inside the \_TodoListState class:

In our case, it'll wait for the user to tap the Add or Cancel button.

The \_displayDialog method will return the showDialog method by building the UI.

The barrierDismissible property is used to define if the pop up has to be closed if the user taps outside of the alert dialog. We have set that to false which means the alert dialog will not be closed on taping outside.

The builder of this showDialog method returns an AlertDialog consisting of title, content, and actions property. The title is set to display the text "Add a todo". The content property will render an text input field with automatic focus enabled and the hint "Type your todo".

The actions property will render 2 buttons, Cancel and Add. The Cancel button is an outlined button, and pressing it will close the dialog. The Add button adds the text to the todo list and closes the dialog.

**How to List the Todos**

We have added the code to add todos to the list. But wait – how can we verify that? We have to find if the todo has actually been added to the list.

Let's verify that by rendering the list of todo items in the UI. To do so, we have to design the UI for a single todo. Let's do that.

Add the following code at the end of main.dart file:





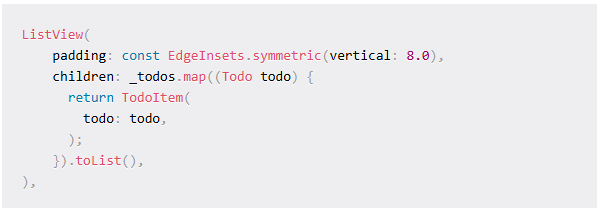
First, we created a class with the TodoItem and we extended it from the StatelessWidget class as we don't need to maintain state for this class.

We accept a Todo, which is passed via constructor to our class. The code in the build method determines the UI. It renders the ListTile widget with the Checkbox widget passed to the leading property.

The title property renders a row of Text and IconButton widgets. The Text widget shows the name of the todo and the IconButton widget displays the delete icon.

Notice the \_getTextStyle method passed to the style property of the Text widget. This method strikes out the text if the todo is marked as complete. Nothing changes on tapping any of these widgets, as the corresponding properties are left empty (onTap, onChanged, and onPressed).

Change the body property of the build method in \_TodoListState with the following code:



The above code defines a ListView widget iterating over the created todos and passing each todo to the TodoItem widget.

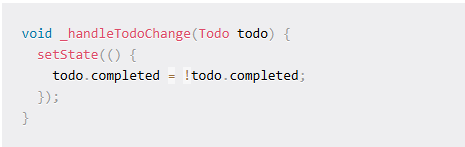
We're done with listing the todos. Let's verify if both creating and viewing a todo works fine.

How to Update a Todo

Let's mark the todo as complete on pressing the checkbox near each todo.

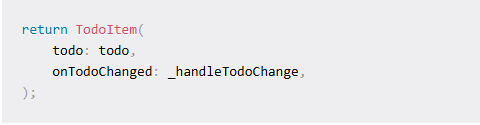
We have 2 fields in our Todo class. They're name and completed status. Whenever a Todo is created, the default value of the completed field is set to false. This means the todo is in progress. We can change that to true whenever we complete the task.

Define a method called \_handleTodoChange in the \_TodoListState class. Add this method below the \_addTodoItem method which we defined to add a todo to the list.

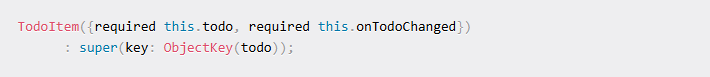


In the above code, we accept a todo and change the completed status of the todo. So, whenever this method is called with a todo, it's completed status will change from true to false or vice versa. Remember that we have wrapped this inside a setState method to render the UI after making the change.

We have to trigger this method when a user taps on a todo or taps on a checkbox. We should pass this method to the TodoItem class. While calling the TodoItem in the build method of the \_TodoListState class, pass the \_handleTodoChange method as shown below:



As we're passing the method to the TodoItem class, we should receive the same method in the TodoItem class. To do so, we have to define this method in the constructor of the TodoItem class. Go to TodoItem and change the constructor to include the onTodoChanged method.

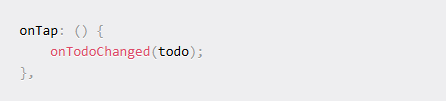


You may notice in the above code that we use \*\*this\*\*.onTodoChanged, which means we're binding the method passed to a method in this TodoItem class.

Let's define a method with the same name and set the return type to void (as we don't expect anything from that method).



So, wherever we call this method in our code, the status of our todo will be changed to the opposite. Let's call this method in the onTap property of the ListTile widget and onChanged property of the Checkbox widget.



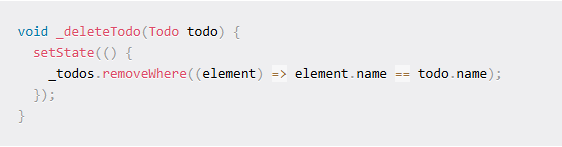
How to Delete a Todo

We have only one item left to complete this app. We should be able to delete a todo, if we create one by mistake or if it's no longer applicable.

Steps to delete a todo are similar to updating a todo. We'll doing the exact 4 steps as we did for updating a todo.

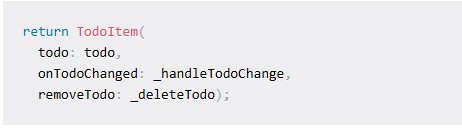
1. Define the \_deleteTodo method
2. Pass the method on TodoItem render
3. Receive the method on TodoItem constructor
4. Bind the method
5. Call the method on button tap

Here's the method to delete the todo. Add this in the \_TodoListState class below the \_handleTodoChange method:

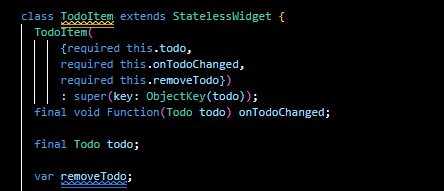


This method accepts a todo, compares it with the todo list, and identifies the todo which matches with this name. Then it deletes it from the list and finally updates the state.

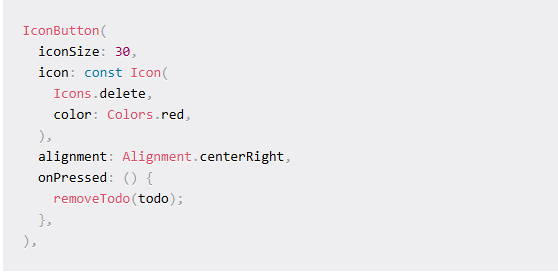
Let's pass the method reference to TodoItem in the build method of the \_TodoListState class.



Change the constructor to accept the removeTodo method.



Our final step is to call this method on pressing the delete button.



**Assessment:**

Create a todo app as we practice in lab. Design a UI like this picture.

