

```
// main content
    Expanded(
        child: Container(
            margin: EdgeInsets.only(left: 16),
            padding: EdgeInsets.symmetric(horizontal: 24,
vertical: 32),
            child: ConstrainedBox(
                constraints: BoxConstraints(maxWidth: 1200),
                child: _isLoading
                    ? Center(child: CircularProgressIndicator())
                    : Column(
                        children: [
                            // title
                            Column(
                                children: [
                                    Column(
                                        children: [
                                            Text(
                                                "Payment Form Review",
                                                style: TextStyle(
                                                    fontSize: 28,
                                                    fontWeight: FontWeight.bold,
                                                ),
                                            ),
                                            Text(
                                                "Review and validate/invalidate
pending payment submissions.",
                                                style: TextStyle(color: Colors.grey,
fontSize: 14),
                                            ),
                                        ],
                                    ),
                                    Spacer(),
                                ],
                            ),
                            Row(
                                children: [
                                    buildStatus(filteredPayments.length),
```

11

SizedBox(height: 24),

```
// table  
paymentsTable(),  
SizedBox(height: 24),  
],
```

whats wrong with this

Good catch — the issue here is with your **layout structure**, specifically this part inside the Column :

dart

 Copy code

```
Column(  
  children: [  
    Column(  
      children: [  
        Text(...),  
        Text(...),  
      ],  
    ),  
    Spacer(), // ✘ Problem here  
  Row(  
    children: [  
      buildStatus(filteredPayme  
    ],  
  ),  
  ],  
,
```

## ✗ The Problem:

You're using a `Spacer()` inside a `Column` that's itself inside another `Column` — but the **parent Column is unbounded in height** because of how it's nested inside `Expanded` → `Container` → `ConstrainedBox`.

In Flutter, a `Spacer()` (and any widget using `Expanded` or `Flexible`) only works inside a `Flex` widget (like a `Row` or `Column`) **that has a bounded main axis**.

Since the main `Column` is trying to expand vertically inside an unbounded height context, Flutter throws this kind of layout error:

css

 Copy code

```
RenderFlex children have non-zero flex but incoming height constraints are unbound
```

## ✓ How to Fix It:

You have a few options depending on what layout behavior you actually want.

### Option 1: Remove `Spacer()`

If you just wanted some spacing between the title and the filter row, replace it with a `SizedBox(height: ...)`:

dart

 Copy code

```
Column(  
  children: [  
    Column(  
      children: [  
        Text(  
          "Payment Form Review",  
          style: TextStyle(fontSize: 28, fontWeight: FontWeight.bold),  
        ),  
        Text(  
          "Review and validate/invalidate pending payment submissions.",  
          style: TextStyle(color: Colors.grey, fontSize: 14),  
        ),  
      ],
```

```
),
    SizedBox(height: 16), // ✓ replaces Spacer()
Row(
  children: [
    buildStatus(filteredPayments.length),
  ],
),
],
),
),
],
```

---

### Option 2: Use MainAxisAlignment.spaceBetween

If you meant to push the filter row to the bottom of that section, you can do this:

dart

 Copy code

```
Column(
  mainAxisAlignment: MainAxisAlignment.spaceBetween, // ✓ handles spacing automatically
  children: [
    Column(
      children: [
        Text(
          "Payment Form Review",
          style: TextStyle(fontSize: 28, fontWeight: FontWeight.bold),
        ),
        Text(
          "Review and validate/invalidate pending payment submissions.",
          style: TextStyle(color: Colors.grey, fontSize: 14),
        ),
      ],
    ),
    Row(
      children: [
        buildStatus(filteredPayments.length),
      ],
    ),
  ],
),
```

### Option 3: Wrap Outer Column in Expanded

If you *really* want `Spacer()` to work, wrap the **outer** Column that contains it in an `Expanded`:

dart

 Copy code

```
Expanded(  
    child: Column(  
        children: [  
            // ... your Spacer() etc.  
        ],  
    ),  
)
```

But this only makes sense if you want the column to fill the available vertical space, which doesn't seem necessary here.

---

#### In short:

Replace `Spacer()` with `SizedBox(height: something)` — that's the simplest and correct fix for your current layout.



```
: Column( children: [  
    Row(  
        mainAxisAlignment:  
            MainAxisAlignment.spaceBetween,  
        children: [  
            // title  
            Column(  
                crossAxisAlignment:  
                    CrossAxisAlignment.start,  
                children: [  
                    Text(  
                        "Title",  
                        style: TextStyle(  
                            color: Colors.purple,  
                            fontSize: 18,  
                            fontWeight: FontWeight.w500),  
                    ),  
                    Text(  
                        "Description",  
                        style: TextStyle(  
                            color: Colors.grey,  
                            fontSize: 14,  
                            fontWeight: FontWeight.w400),  
                    ),  
                ],  
            ),  
        ],  
    ),  
    Container(  
        height: 100,  
        width: 100,  
        decoration: BoxDecoration(  
            shape: BoxShape.circle,  
            color: Colors.purple,  
        ),  
    ),  
],  
)
```

```
"Payment Form Review",
style: TextStyle(
  fontSize: 28,
  fontWeight: FontWeight.bold,
),
),
Text(
  "Review and validate/invalidate
pending payment submissions.",
style: TextStyle(color:
Colors.grey, fontSize: 14),
),
],
),
),
// filter and counter
Row(
  crossAxisAlignment:
CrossAxisAlignment.end,
  children: [
buildStatus(filteredPayments.length),
],
),
],
),
),
SizedBox(height: 24),
// table
Expanded(
  child: Container( child:
paymentsTable(), ),
),
],
),
);
```

my table isn't taking all the horizontal space. what do i do

Your layout is **almost** correct — the issue is inside your **table widget** (or the container around it), not with the Column.

Right now you have:

dart

Copy code

```
Expanded(
    child: Container(
        child: paymentsTable(),
    ),
),
```

But this **does NOT force the table to stretch horizontally unless:**

- The `Container` has a width constraint, **or**
- The table widget itself expands, **or**
- The table is inside a `Row` / `Expanded` combo.

### Fix: Force the table to take full width

Set the container's width to `double.infinity`:

dart

Copy code

```
Expanded(
    child: Container(
        width: double.infinity,           //  forces full horizontal width
        child: paymentsTable(),
    ),
),
```

### If `paymentsTable()` returns a `DataTable`:

Flutter's `DataTable` often shrinks to the minimum width unless wrapped in `FittedBox` or `SingleChildScrollView`.

Use this structure:

dart

Copy code

```
Expanded(  
    child: SingleChildScrollView(  
        scrollDirection: Axis.horizontal,  
        child: ConstrainedBox(  
            constraints: BoxConstraints(minWidth: double.infinity),  
            child: paymentsTable(),  
        ),  
    ),  
)
```

This ensures the table:

- grows to full width
  - stays scrollable horizontally if columns overflow
- 

- If you want the table to always fill width, simplest version is:

dart

 Copy code

```
Expanded(  
    child: SizedBox(  
        width: double.infinity,  
        child: paymentsTable(),  
    ),  
)
```

If you want, paste your `paymentsTable()` code and I can adjust it precisely.

< 1/2 >      ...

```
class DropdownInputField extends StatelessWidget {  
    final String label;  
    final TextEditingController controller;  
    final List<String> items;  
    final String? Function(String?)? validator;
```

```
const DropdownInputField({  
    super.key,  
    required this.label,  
    this.controller,  
    required this.items,  
    this.validator,  
});
```

is there something wrong here

Yes — there's one clear issue in your code 

### Problem:

You declared the parameter `controller` as **required**, but its type is **non-nullable** while also marking it as **optional** (`this.controller`,) — meaning you didn't mark it as required in the constructor parameters list, yet it's non-nullable.

Let's look closely:

dart

 Copy code

```
final TextEditingController controller; //  non-nullable  
  
const DropdownInputField({  
    super.key,  
    required this.label,  
    this.controller, //  not marked as required but can't be null  
    required this.items,  
    this.validator,  
});
```

This causes a **compile-time error**:

The parameter 'controller' can't have a value of 'null' because of its type, but the implicit default value is 'null'.

## Fix Options (depending on what you need):

### Option 1: Make it required

If every dropdown input field must have a controller, mark it as required :

dart

Copy code

```
final TextEditingController controller;  
  
const DropdownInputField({  
    super.key,  
    required this.label,  
    required this.controller, //   
    required this.items,  
    this.validator,  
});
```

Use like:

dart

Copy code

```
DropdownInputField(  
    label: "Category",  
    controller: categoryController,  
    items: ["A", "B", "C"],  
)
```

---

### Option 2: Make it nullable

If some dropdowns don't need a controller, make it nullable:

dart

Copy code

```
final TextEditingController? controller; //  nullable  
  
const DropdownInputField({  
    super.key,  
    required this.label,  
    this.controller, //  optional and nullable  
    required this.items,
```

```
this.validator,  
});
```

Then, in your widget code, use:

dart

 Copy code

```
controller?.text ?? ''
```

where you reference it.

---

### Summary:

Choose **one**:

- required `this.controller` → if all dropdowns must have controllers
- `TextEditingController? controller` → if it's optional

Otherwise, Dart will complain about non-nullable fields not being initialized.

       ...

how do i do the second

Perfect  here's exactly how to do **Option 2 (make controller optional/nullable)**:

---

### Correct version:

dart

 Copy code

```
class DropdownInputField extends StatelessWidget {  
    final String label;  
    final TextEditingController? controller; //  make it nullable  
    final List<String> items;  
    final String? Function(String?)? validator;  
  
    const DropdownInputField({
```

```
super.key,  
    required this.label,  
    this.controller, // ✅ now safely optional  
    required this.items,  
    this.validator,  
);  
  
@override  
Widget build(BuildContext context) {  
    return DropdownButtonFormField<String>(  
        value: controller?.text.isNotEmpty == true ? controller!.text : null, // ✅  
        decoration: InputDecoration(labelText: label),  
        items: items  
        .map((item) => DropdownMenuItem(  
            value: item,  
            child: Text(item),  
        ))  
        .toList(),  
        onChanged: (value) {  
            if (controller != null) {  
                controller!.text = value ?? '';  
            }  
        },  
        validator: validator,  
    );  
}  
}
```

## 🧠 Explanation:

- `TextEditingController? controller` → the field can now be `null`.
- `this.controller` → optional in constructor.
- `controller?.text` → safely checks if controller exists.
- `controller!.text` → force unwrap only after null check.

## ✅ Example usage:

If you want to **use a controller**:

dart

 Copy code

```
final typeController = TextEditingController();  
  
DropdownInputField(  
    label: 'Payment Type',  
    controller: typeController,  
    items: ['GCash', 'Bank', 'Cash'],  
)
```

If you **don't need** a controller:

dart

 Copy code

```
DropdownInputField(  
    label: 'Payment Type',  
    items: ['GCash', 'Bank', 'Cash'],  
)
```

No errors, fully safe 

     ...

View reports for different kinds of report types.

paraphrase

Access and review various types of reports.

     ...

"Active Loans",  
"Overdue Loans",  
"Member Loan Summary",

"Payment Collection",  
"Missed Payments",  
"Voucher & Revenue Summary"

what columns are the tables for these supposed to have

Excellent question — here's a clear, practical breakdown of what **columns** each of those report tables would normally have, based on common financial and lending system designs 

---

## 1. Active Loans

**Purpose:** Show all loans currently being paid (not yet completed or overdue).

**Suggested columns:**

Column	Description
Loan ID	Unique loan reference number
Member Name	Borrower's full name
Loan Type	e.g., Personal, Business, Emergency
Principal Amount	Original loan amount
Remaining Balance	Unpaid amount left
Start Date	Date loan was approved/released
Due Date	Expected end of term
Status	e.g., Active, On Track

---

## 2. Overdue Loans

**Purpose:** Highlight loans with missed or late payments.

**Suggested columns:**

Column	Description
Loan ID	Loan reference
Member Name	Borrower's name
Loan Type	Type of loan
Due Date	Original due date
Days Overdue	How long it has been overdue
Amount Due	Total unpaid amount
Penalty	Accumulated late fees
Contact Number	For follow-up
Status	e.g., Overdue, In Collection

### 3. Member Loan Summary

**Purpose:** Show an overview of each member's total loan activity.

**Suggested columns:**

Column	Description
Member Name	Borrower's name
Member ID	Unique identifier
Total Loans	Number of loans availed
Total Borrowed	Total amount borrowed
Total Paid	Total amount paid back
Outstanding Balance	Remaining unpaid balance
Last Payment Date	Most recent payment

Column	Description
Loan Status	e.g., Active, Cleared

## 4. Payment Collection

**Purpose:** Track all payments received.

**Suggested columns:**

Column	Description
Payment ID	Unique payment reference
Member Name	Payer's name
Loan ID	Associated loan
Payment Date	When payment was made
Payment Method	e.g., GCash, Bank Transfer, Cash
Amount Paid	Amount of this transaction
Collected By	Staff who received payment
Remarks	Optional notes

## 5. Missed Payments

**Purpose:** Show scheduled payments that were not made on time.

**Suggested columns:**

Column	Description
Loan ID	Related loan
Member Name	Borrower's name
Due Date	Expected payment date

Column	Description
Amount Due	Amount expected
Days Missed	Number of days overdue
Contact Number	For follow-up
Next Payment Date	Next scheduled payment
Status	e.g., Missed, Settled Later

## 6. Voucher & Revenue Summary

**Purpose:** Show income from fees, interest, and voucher-based transactions.

**Suggested columns:**

Column	Description
Voucher ID	Transaction or voucher reference
Date Issued	When the voucher/revenue was recorded
Description	What the voucher was for (e.g., Interest Payment, Service Fee)
Amount	Amount earned
Revenue Type	e.g., Interest, Processing Fee, Penalty
Recorded By	Staff who entered the record
Remarks	Optional notes

Would you like me to make the **Flutter DataTable column definitions** (with column headers and example data structure) for each of these too? It'll make it easier to plug straight into your UI.



how do i edit this into a function with dynamic parameters

```
void onSort(int columnIndex, bool ascending) {  
    setState(() {  
        sortColumnIndex = columnIndex;  
        isAscending = ascending;  
  
        switch (columnIndex) {  
            case 0:  
                loans.sort((a, b) => ascending  
                    ? a["ref"].compareTo(b["ref"])  
                    : b["ref"].compareTo(a["ref"]));  
                break;  
            case 1:  
                loans.sort((a, b) => ascending  
                    ? a["amt"].compareTo(b["amt"])  
                    : b["amt"].compareTo(a["amt"]));  
                break;  
            case 2:  
                loans.sort((a, b) => ascending  
                    ? a["interest"].compareTo(b["interest"])  
                    : b["interest"].compareTo(a["interest"]));  
                break;  
            case 3:  
                loans.sort((a, b) => ascending  
                    ? a["start"].compareTo(b["start"])  
                    : b["start"].compareTo(a["start"]));  
                break;  
            case 4:  
                loans.sort((a, b) => ascending  
                    ? a["due"].compareTo(b["due"])  
                    : b["due"].compareTo(a["due"]));  
                break;  
            case 5:  
                loans.sort((a, b) => ascending  
                    ? a["instType"].compareTo(b["instType"])  
                    : b["instType"].compareTo(a["instType"]));  
                break;  
        }  
    });  
}
```

```
case 6:  
    loans.sort((a, b) => ascending  
        ? a["totalInst"].compareTo(b["totalInst"])  
        : b["totalInst"].compareTo(a["totalInst"]));  
    break;  
case 7:  
    loans.sort((a, b) => ascending  
        ? a["instAmt"].compareTo(b["instAmt"])  
        : b["instAmt"].compareTo(a["instAmt"]));  
    break;  
case 8:  
    loans.sort((a, b) => ascending  
        ? a["status"].compareTo(b["status"])  
        : b["status"].compareTo(a["status"]));  
    break;  
}  
});  
}
```

```
Widget amortizationTable() {  
    return Expanded(  
        child: Container(  
            padding: EdgeInsets.all(16),  
            decoration: BoxDecoration(  
                color: Colors.white,  
                borderRadius: BorderRadius.circular(8),  
                boxShadow: [  
                    BoxShadow(  
                        color: Colors.black.withOpacity(0.05),  
                        blurRadius: 8,  
                        offset: Offset(0, 2),  
                    ),  
                ],  
            ),  
            child: LayoutBuilder(  
                builder: (context, constraints) {  
                    return SingleChildScrollView(  
                        scrollDirection: Axis.vertical,  
                        child: SingleChildScrollView(  
                            scrollDirection: Axis.horizontal,
```

```
child: ConstrainedBox(  
    constraints: BoxConstraints(  
        minWidth: constraints maxWidth,  
    ),  
    child: DataTable(  
        sortColumnIndex: sortColumnIndex,  
        sortAscending: isAscending,  
        columnSpacing: 58,  
        columns: [  
            DataColumn(  
                label: Text("Ref. No.", style:  
                    TextStyle(fontWeight: FontWeight.bold)),  
                onSort: (i, asc) => onSort(i, asc)),  
            DataColumn(  
                label: Text("Amt.", style:  
                    TextStyle(fontWeight: FontWeight.bold)),  
                numeric: true,  
                onSort: (i, asc) => onSort(i, asc)),  
            DataColumn(  
                label: Text("Interest", style:  
                    TextStyle(fontWeight: FontWeight.bold)),  
                numeric: true,  
                onSort: (i, asc) => onSort(i, asc)),  
            DataColumn(  
                label: Text("Start Date", style:  
                    TextStyle(fontWeight: FontWeight.bold)),  
                onSort: (i, asc) => onSort(i, asc)),  
            DataColumn(  
                label: Text("Due Date", style:  
                    TextStyle(fontWeight: FontWeight.bold)),  
                onSort: (i, asc) => onSort(i, asc)),  
            DataColumn(  
                label: Text("Inst Type", style:  
                    TextStyle(fontWeight: FontWeight.bold)),  
                onSort: (i, asc) => onSort(i, asc)),  
            DataColumn(  
                label: Text("Total Inst", style:  
                    TextStyle(fontWeight: FontWeight.bold)),  
                numeric: true,  
                onSort: (i, asc) => onSort(i, asc)),  
        ]  
    )  
);
```

```
    DataColumn(  
        label: Text("Inst Amt.", style:  
            TextStyle(fontWeight: FontWeight.bold)),  
        numeric: true,  
        onSort: (i, asc) => onSort(i, asc)),  
    DataColumn(  
        label: Text("Status", style:  
            TextStyle(fontWeight: FontWeight.bold)),  
        onSort: (i, asc) => onSort(i, asc)),  
    ],  
    rows: loans  
.map(  
    (loan) => DataRow(cells: [  
        DataCell(Text(loan["ref"])),  
        DataCell(Text("P${loan["amt"]}")),  
        DataCell(Text("${loan["interest"]}%")),  
        DataCell(Text(loan["start"])),  
        DataCell(Text(loan["due"])),  
        DataCell(Text(loan["instType"])),  
        DataCell(Text("${loan["totalInst"]}")),  
        DataCell(Text("P${loan["instAmt"]}")),  
        DataCell(Text(loan["status"])),  
    ]),  
)  
.toList(),  
,  
,  
,  
);  
,  
,  
);  
}  
}
```

Perfect — you can **make your `onSort` function dynamic** instead of hardcoding every case. This will make it reusable for **any table, any column, and any data key**.

Here's how to refactor it 

## ✓ Refactored Dynamic Sort Function

dart

 Copy code

```
void onSort<T>(
    int columnIndex,
    bool ascending,
    List<Map<String, dynamic>> data,
    String key,
) {
    setState(() {
        sortColumnIndex = columnIndex;
        isAscending = ascending;

        data.sort((a, b) {
            final valueA = a[key];
            final valueB = b[key];

            // Handle null safety
            if (valueA == null || valueB == null) return 0;

            // Determine comparison type
            if (valueA is num && valueB is num) {
                return ascending ? valueA.compareTo(valueB) : valueB.compareTo(valueA);
            } else if (valueA is DateTime && valueB is DateTime) {
                return ascending ? valueA.compareTo(valueB) : valueB.compareTo(valueA);
            } else {
                return ascending
                    ? valueA.toString().compareTo(valueB.toString())
                    : valueB.toString().compareTo(valueA.toString());
            }
        });
    });
}
```

## ✓ How to Use It in Your DataTable

Instead of calling:

dart

 Copy code

```
onSort: (i, asc) => onSort(i, asc),
```

You now pass the **data list** (`loans`) and the **key name** for that column.

Example:

dart

 Copy code

```
DataColumn(
  label: Text("Ref. No.", style: TextStyle(fontWeight: FontWeight.bold)),
  onSort: (i, asc) => onSort(i, asc, loans, "ref"),
),
DataColumn(
  label: Text("Amt.", style: TextStyle(fontWeight: FontWeight.bold)),
  numeric: true,
  onSort: (i, asc) => onSort(i, asc, loans, "amt"),
),
DataColumn(
  label: Text("Interest", style: TextStyle(fontWeight: FontWeight.bold)),
  numeric: true,
  onSort: (i, asc) => onSort(i, asc, loans, "interest"),
),
DataColumn(
  label: Text("Start Date", style: TextStyle(fontWeight: FontWeight.bold)),
  onSort: (i, asc) => onSort(i, asc, loans, "start"),
),
...

```

## ⚡ Bonus (Date Strings)

If your dates (`start`, `due`) are stored as **strings**, you can enhance the sort function to automatically detect and parse them:

dart

 Copy code

```
DateTime? tryParseDate(value) {
  try {
```

```

        return DateTime.parse(value.toString());
    } catch (_) {
        return null;
    }
}

```

Then update inside the comparator:

dart

 Copy code

```

final dateA = tryParseDate(valueA);
final dateB = tryParseDate(valueB);
if (dateA != null && dateB != null) {
    return ascending ? dateA.compareTo(dateB) : dateB.compareTo(dateA);
}

```

---

## Summary

You now have a **single universal sorter**:

- Works for **any list of maps**
- Handles **strings, numbers, and dates**
- No more giant switch statements 

Would you like me to show how to extend this so it supports *different lists* (e.g., active loans, overdue loans, payments, etc.) dynamically too?



```

void onSort<T>(
    int columnIndex,
    bool ascending,
    List<Map<String, dynamic>> data,
    String key,
) {
    setState(() {
        sortColumnIndex = columnIndex;
        isAscending = ascending;
    });
}

```

```
data.sort((a, b) {
    final valueA = a[key];
    final valueB = b[key];

    // Handle null safety
    if (valueA == null || valueB == null) return 0;

    // Determine comparison type
    if (valueA is num && valueB is num) {
        return ascending ? valueA.compareTo(valueB) :
            valueB.compareTo(valueA);
    } else if (valueA is DateTime && valueB is DateTime) {
        return ascending ? valueA.compareTo(valueB) :
            valueB.compareTo(valueA);
    } else {
        return ascending
            ? valueA.toString().compareTo(valueB.toString())
            : valueB.toString().compareTo(valueA.toString());
    }
});

});

});

}

Widget activeLoansTable(List<Map<String, dynamic>>
loans) {
    return Expanded(
        child: Container(
            padding: EdgeInsets.all(16),
            decoration: BoxDecoration(
                color: Colors.white,
                borderRadius: BorderRadius.circular(8),
                boxShadow: [
                    BoxShadow(
                        color: Colors.black.withOpacity(0.05),
                        blurRadius: 8,
                        offset: Offset(0, 2),
                    ),
                ],
            ),
        ),
    );
}
```

```
child: LayoutBuilder(  
    builder: (context, constraints) {  
        return SingleChildScrollView(  
            scrollDirection: Axis.vertical,  
            child: SingleChildScrollView(  
                scrollDirection: Axis.horizontal,  
                child: ConstrainedBox(  
                    constraints: BoxConstraints(  
                        minWidth: constraints maxWidth,  
                    ),  
                    child: DataTable(  
                        sortColumnIndex: sortColumnIndex,  
                        sortAscending: isAscending,  
                        columnSpacing: 58,  
                        columns: [  
                            DataColumn(  
                                label: Text("Loan ID", style:  
                                    TextStyle(fontWeight: FontWeight.bold)),  
                                onSort: (i, asc) => onSort(i, asc)),  
                            DataColumn(  
                                label: Text("Member Name", style:  
                                    TextStyle(fontWeight: FontWeight.bold)),  
                                numeric: true,  
                                onSort: (i, asc) => onSort(i, asc)),  
                            DataColumn(  
                                label: Text("Loan Type", style:  
                                    TextStyle(fontWeight: FontWeight.bold)),  
                                numeric: true,  
                                onSort: (i, asc) => onSort(i, asc)),  
                            DataColumn(  
                                label: Text("Principal Amount", style:  
                                    TextStyle(fontWeight: FontWeight.bold)),  
                                onSort: (i, asc) => onSort(i, asc)),  
                            DataColumn(  
                                label: Text("Remaining Balance", style:  
                                    TextStyle(fontWeight: FontWeight.bold)),  
                                onSort: (i, asc) => onSort(i, asc)),  
                            DataColumn(  
                                label: Text("Start Date", style:  
                                    TextStyle(fontWeight: FontWeight.bold)),
```

```
onSort: (i, asc) => onSort(i, asc)),  
    DataColumn(  
        label: Text("Due Date", style:  
            TextStyle(fontWeight: FontWeight.bold)),  
        numeric: true,  
        onSort: (i, asc) => onSort(i, asc)),  
    ],  
    rows: loans  
    .map(  
        (loan) => DataRow(cells: [  
            DataCell(Text(loan["loanID"])),  
            DataCell(Text("P${loan["memName"]}")),  
            DataCell(Text("${loan["loanType"]}%")),  
            DataCell(Text(loan["principalAmt"])),  
            DataCell(Text(loan["remainBal"])),  
            DataCell(Text(loan["startDate"])),  
            DataCell(Text("${loan["dueDate"]}")),  
        ]),  
    )  
    .toList(),  
),  
,  
,  
);  
},  
),  
),  
);  
}  
}
```

does this look right

You're very close! 🔥

Structurally, your function and table setup are both solid — but there are a few **minor mistakes and improvements** worth fixing to make it actually work and render properly.

Let's go through them clearly ⤵

## ✖ Issues and Fixes

### 1 onSort Calls Missing Parameters

In your `DataColumn`s, you wrote:

dart

 Copy code

```
onSort: (i, asc) => onSort(i, asc),
```

But your `onSort` function expects **four parameters**:

dart

 Copy code

```
onSort<T>(int columnIndex, bool ascending, List<Map<String, dynamic>> data, String
```

So, you need to pass the `loans` list and the column key (the map key for that column).

#### ✓ Fix:

dart

 Copy code

```
onSort: (i, asc) => onSort(i, asc, loans, "loanID"),
```

and so on for each column.

---

### 2 Wrong numeric Flags

Only number-based columns should have `numeric: true`.

"Member Name" and "Loan Type" shouldn't be numeric.

#### ✓ Fix numeric usage:

dart

 Copy code

```
DataColumn(label: Text("Loan ID", style: boldStyle), onSort: (i, asc) => onSort(i, asc), numeric: false),
DataColumn(label: Text("Member Name", style: boldStyle), onSort: (i, asc) => onSort(i, asc), numeric: false),
DataColumn(label: Text("Loan Type", style: boldStyle), onSort: (i, asc) => onSort(i, asc), numeric: false),
DataColumn(label: Text("Principal Amount", style: boldStyle), numeric: true, onSort: (i, asc) => onSort(i, asc)),
DataColumn(label: Text("Remaining Balance", style: boldStyle), numeric: true, onSort: (i, asc) => onSort(i, asc))
```

```
 DataColumn(label: Text("Start Date", style: boldStyle), onSort: (i, asc) => onSort(i, asc)),
 DataColumn(label: Text("Due Date", style: boldStyle), onSort: (i, asc) => onSort(i, asc)),
```

### 3 Wrong “₱” and “%” Placements

You have:

dart

Copy code

```
DataCell(Text("₱${loan["memName"]}")),
DataCell(Text("${loan["loanType"]}%")),
```

Those symbols don't belong there — looks like a copy-paste mistake from your old code.

#### Fix:

dart

Copy code

```
DataCell(Text(loan["memName"])),
DataCell(Text(loan["loanType"])),
DataCell(Text("₱${loan["principalAmt"]}")),
DataCell(Text("₱${loan["remainBal"]}")),
```

### 4 (Optional) Style Constant for Clean Code

You can define this for readability:

dart

Copy code

```
final boldStyle = const TextStyle(fontWeight: FontWeight.bold);
```

### Final Clean Version

dart

Copy code

```
void onSort<T>(
    int columnIndex,
    bool ascending,
    List<Map<String, dynamic>> data,
    String key,
) {
    setState(() {
        sortColumnIndex = columnIndex;
        isAscending = ascending;

        data.sort((a, b) {
            final valueA = a[key];
            final valueB = b[key];

            if (valueA == null || valueB == null) return 0;

            if (valueA is num && valueB is num) {
                return ascending ? valueA.compareTo(valueB) : valueB.compareTo(valueA);
            } else if (valueA is DateTime && valueB is DateTime) {
                return ascending ? valueA.compareTo(valueB) : valueB.compareTo(valueA);
            } else {
                return ascending
                    ? valueA.toString().compareTo(valueB.toString())
                    : valueB.toString().compareTo(valueA.toString());
            }
        });
    });
}

Widget activeLoansTable(List<Map<String, dynamic>> loans) {
    const boldStyle = TextStyle(fontWeight: FontWeight.bold);

    return Expanded(
        child: Container(
            padding: EdgeInsets.all(16),
            decoration: BoxDecoration(
                color: Colors.white,
                borderRadius: BorderRadius.circular(8),
                boxShadow: [
                    BoxShadow(

```

```
color: Colors.black.withOpacity(0.05),  
blurRadius: 8,  
offset: Offset(0, 2),  
,  
],  
,  
child: LayoutBuilder(  
    builder: (context, constraints) {  
        return SingleChildScrollView(  
            scrollDirection: Axis.vertical,  
            child: SingleChildScrollView(  
                scrollDirection: Axis.horizontal,  
                child: ConstrainedBox(  
                    constraints: BoxConstraints(minWidth: constraints maxWidth),  
                    child: DataTable(  
                        sortColumnIndex: sortColumnIndex,  
                        sortAscending: isAscending,  
                        columnSpacing: 58,  
                        columns: [  
                            DataColumn(label: Text("Loan ID", style: boldStyle), onSort:  
                            DataColumn(label: Text("Member Name", style: boldStyle), onSo  
                            DataColumn(label: Text("Loan Type", style: boldStyle), onSort  
                            DataColumn(label: Text("Principal Amount", style: boldStyle),  
                            DataColumn(label: Text("Remaining Balance", style: boldStyle)  
                            DataColumn(label: Text("Start Date", style: boldStyle), onSor  
                            DataColumn(label: Text("Due Date", style: boldStyle), onSort:  
                        ],  
                        rows: loans.map((loan) {  
                            return DataRow(cells: [  
                                DataCell(Text(loan["loanID"].toString())),  
                                DataCell(Text(loan["memName"])),  
                                DataCell(Text(loan["loanType"])),  
                                DataCell(Text("P${loan["principalAmt"]}")),  
                                DataCell(Text("P${loan["remainBal"]}")),  
                                DataCell(Text(loan["startDate"])),  
                                DataCell(Text(loan["dueDate"])),  
                            ]);  
                        }).toList(),  
                    ),  
                ),  
            ),  
        );  
    },  
);
```

```
    ),  
    );  
},  
),  
);  
};
```

## ✓ Summary

- ✓ Uses your new dynamic `onSort()` correctly
- ✓ Fixes misplaced symbols and numeric flags
- ✓ Clean, scalable structure — can reuse this for other tables easily

Would you like me to show how to **reuse the same table builder** for "Overdue Loans," "Payment Collection," etc., just by passing in different column definitions and data?



```
import 'package:flutter/material.dart';  
import 'package:the_basics/widgets/top_navbar.dart';  
import 'package:the_basics/widgets/side_menu.dart';  
import 'package:the_basics/widgets/input_fields.dart';  
import 'package:supabase_flutter/supabase_flutter.dart';  
  
class AdminRecords extends StatefulWidget {  
  const AdminRecords({super.key});  
  
  @override  
  State<AdminRecords> createState() =>  
  _AdminRecordsState();  
}  
  
class _AdminRecordsState extends State<AdminRecords> {  
  int? sortColumnIndex;
```

```
bool isAscending = true;
```

```
Widget buttonsAndFiltersRow() {
    return Row(
        children: [
            SizedBox(
                width: 250,
                child: DropdownInputField(
                    label: "Report Type",
                    items: [
                        "Active Loans",
                        "Overdue Loans",
                        "Member Loan Summary",
                        "Payment Collection",
                        "Missed Payments",
                        "Voucher & Revenue Summary"
                    ],
                ),
            ),
            SizedBox(width: 16),
        ],
    );
}
```

```
SizedBox(
    width: 250,
    child: DropdownInputField(
        label: "Filter by...",
        items: [
            "Month",
            "Quarter",
            "Year",
        ],
    ),
),
SizedBox(width: 16),
```

```
SizedBox(
    height: 28,
    child: ElevatedButton.icon(
        onPressed: () {},
        label: const Text(
            "Generate Report",

```

```
        style: TextStyle(color: Colors.white),
    ),
    style: ElevatedButton.styleFrom(
        backgroundColor: Colors.black,
        minimumSize: const Size(100, 28),
        padding:
            const EdgeInsets.symmetric(horizontal: 8),
    ),
),
),
),
),

Spacer(),

SizedBox(
    height: 28,
    child: ElevatedButton.icon(
        onPressed: () {},
        icon: const Icon(Icons.download,
            color: Colors.white),
        label: const Text(
            "Download",
            style: TextStyle(color: Colors.white),
        ),
        style: ElevatedButton.styleFrom(
            backgroundColor: Colors.black,
            minimumSize: const Size(100, 28),
            padding:
                const EdgeInsets.symmetric(horizontal: 8),
        ),
),
),
),
),
],
);
}
}

void onSort<T>(
    int columnIndex,
    bool ascending,
    List<Map<String, dynamic>> data,
    String key,
```

```
) {  
    setState(() {  
        sortColumnIndex = columnIndex;  
        isAscending = ascending;  
  
        data.sort((a, b) {  
            final valueA = a[key];  
            final valueB = b[key];  
  
            if (valueA == null || valueB == null) return 0;  
  
            if (valueA is num && valueB is num) {  
                return ascending ? valueA.compareTo(valueB) :  
                    valueB.compareTo(valueA);  
            } else if (valueA is DateTime && valueB is DateTime) {  
                return ascending ? valueA.compareTo(valueB) :  
                    valueB.compareTo(valueA);  
            } else {  
                return ascending  
                    ? valueA.toString().compareTo(valueB.toString())  
                    : valueB.toString().compareTo(valueA.toString());  
            }  
        });  
    });  
}  
  
// all loans currently being paid (not yet completed or  
// overdue).  
Widget activeLoansTable(List<Map<String, dynamic>>  
loans) {  
    const boldStyle = TextStyle(fontWeight:  
        FontWeight.bold);  
  
    return Expanded(  
        child: Container(  
            padding: EdgeInsets.all(16),  
            decoration: BoxDecoration(  
                color: Colors.white,  
                borderRadius: BorderRadius.circular(8),  
                boxShadow: [
```

```
BoxShadow(  
    color: Colors.black.withOpacity(0.05),  
    blurRadius: 8,  
    offset: Offset(0, 2),  
,  
],  
,  
child: LayoutBuilder(  
    builder: (context, constraints) {  
        return SingleChildScrollView(  
            scrollDirection: Axis.vertical,  
            child: SingleChildScrollView(  
                scrollDirection: Axis.horizontal,  
                child: ConstrainedBox(  
                    constraints: BoxConstraints(minWidth:  
constraints.maxWidth),  
                    child: DataTable(  
                        sortColumnIndex: sortColumnIndex,  
                        sortAscending: isAscending,  
                        columnSpacing: 58,  
                        columns: [  
                            DataColumn(label: Text("Loan ID", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"loanID")),  
                            DataColumn(label: Text("Member Name",  
style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"memName")),  
                            DataColumn(label: Text("Loan Type", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"loanType")),  
                            DataColumn(label: Text("Principal Amount",  
style: boldStyle), numeric: true, onSort: (i, asc) => onSort(i,  
asc, loans, "principalAmt")),  
                            DataColumn(label: Text("Remaining Balance",  
style: boldStyle), numeric: true, onSort: (i, asc) => onSort(i,  
asc, loans, "remainBal")),  
                            DataColumn(label: Text("Start Date", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"startDate")),  
                            DataColumn(label: Text("Due Date", style:
```

```
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,
"dueDate")),
],
rows: loans.map((loan) {
    return DataRow(cells: [
        DataCell(Text(loan["loanID"].toString())),
        DataCell(Text(loan["memName"])),
        DataCell(Text(loan["loanType"])),
        DataCell(Text("P${loan["principalAmt"]}")),
        DataCell(Text("P${loan["remainBal"]}")),
        DataCell(Text(loan["startDate"])),
        DataCell(Text(loan["dueDate"])),
    ]);
}),
),
),
),
);
},
),
),
),
);
}
}

// loans with missed or late payments.
Widget overdueLoansTable(List<Map<String, dynamic>>
loans) {
    const boldStyle = TextStyle(fontWeight:
FontWeight.bold);

    return Expanded(
        child: Container(
            padding: EdgeInsets.all(16),
            decoration: BoxDecoration(
                color: Colors.white,
                borderRadius: BorderRadius.circular(8),
                boxShadow: [
                    BoxShadow(
                        color: Colors.black.withOpacity(0.05),
                        blurRadius: 8,
```

```
        offset: Offset(0, 2),  
    ),  
],  
,  
child: LayoutBuilder(  
    builder: (context, constraints) {  
        return SingleChildScrollView(  
            scrollDirection: Axis.vertical,  
            child: SingleChildScrollView(  
                scrollDirection: Axis.horizontal,  
                child: ConstrainedBox(  
                    constraints: BoxConstraints(minWidth:  
constraints.maxWidth),  
                    child: DataTable(  
                        sortColumnIndex: sortColumnIndex,  
                        sortAscending: isAscending,  
                        columnSpacing: 58,  
                        columns: [  
                            DataColumn(label: Text("Loan ID", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"loanID")),  
                            DataColumn(label: Text("Member Name",  
style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"memName")),  
                            DataColumn(label: Text("Loan Type", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"loanType")),  
                            DataColumn(label: Text("Due Date", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"dueDate")),  
                            DataColumn(label: Text("Days Overdue",  
style: boldStyle), numeric: true, onSort: (i, asc) => onSort(i,  
asc, loans, "daysOverdue")),  
                            DataColumn(label: Text("Amount Due", style:  
boldStyle), numeric: true, onSort: (i, asc) => onSort(i, asc,  
loans, "amountDue")),  
                            DataColumn(label: Text("Accumulated Late  
Fees", style: boldStyle), numeric: true, onSort: (i, asc) =>  
onSort(i, asc, loans, "lateFees")),  
                            DataColumn(label: Text("Contact No.", style:
```

```
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,
"contactNo")),
],
rows: loans.map((loan) {
return DataRow(cells: [
DataCell(Text(loan["loanID"].toString())),
DataCell(Text(loan["memName"])),
DataCell(Text(loan["loanType"])),
DataCell(Text(loan["dueDate"]),
DataCell(Text(loan["daysOverdue"].toString())),
DataCell(Text("P${loan["amountDue"]}")),
DataCell(Text("P${loan["lateFees"]}")),
DataCell(Text(loan["contactNo"])),
]);
}),
),
),
),
),
);
},
),
),
),
);
}
}

// each member's total loan activity.
Widget memberLoansTable(List<Map<String, dynamic>>
loans) {
const boldStyle = TextStyle(fontWeight:
FontWeight.bold);

return Expanded(
child: Container(
padding: EdgeInsets.all(16),
decoration: BoxDecoration(
color: Colors.white,
borderRadius: BorderRadius.circular(8),
boxShadow: [
BoxShadow(

```

```
color: Colors.black.withOpacity(0.05),  
blurRadius: 8,  
offset: Offset(0, 2),  
,  
],  
,  
child: LayoutBuilder(  
builder: (context, constraints) {  
return SingleChildScrollView(  
scrollDirection: Axis.vertical,  
child: SingleChildScrollView(  
scrollDirection: Axis.horizontal,  
child: ConstrainedBox(  
constraints: BoxConstraints(minWidth:  
constraints.maxWidth),  
child: DataTable(  
sortColumnIndex: sortColumnIndex,  
sortAscending: isAscending,  
columnSpacing: 58,  
columns: [  
DataColumn(label: Text("Member Name",  
style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"memName")),  
DataColumn(label: Text("Member ID", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"memID")),  
DataColumn(label: Text("Total Loans", style:  
boldStyle), numeric: true, onSort: (i, asc) => onSort(i, asc,  
loans, "totalLoans")),  
DataColumn(label: Text("Total Borrowed",  
style: boldStyle), numeric: true, onSort: (i, asc) => onSort(i,  
asc, loans, "totalBorrowed")),  
DataColumn(label: Text("Total Paid", style:  
boldStyle), numeric: true, onSort: (i, asc) => onSort(i, asc,  
loans, "totalPaid")),  
DataColumn(label: Text("Outstanding  
Balance", style: boldStyle), numeric: true, onSort: (i, asc) =>  
onSort(i, asc, loans, "outBal")),  
DataColumn(label: Text("Last Payment Date",  
style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,
```

```
"lastPaid")),
        DataColumn(label: Text("Loan Status", style:
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,
"loanStatus")),
    ],
rows: loans.map((loan) {
    return DataRow(cells: [
        DataCell(Text(loan["memName"])),
        DataCell(Text(loan["memID"].toString())),
        DataCell(Text("P${loan["totalLoans"]}")),
        DataCell(Text("P${loan["totalBorrowed"]}")),
        DataCell(Text("P${loan["totalPaid"]}")),
        DataCell(Text("P${loan["outBal"]}")),
        DataCell(Text(loan["lastPaid"])),
        DataCell(Text(loan["loanStatus"])),
    ]);
}),
),
),
),
);
},
),
),
),
),
);
}
}

// Track all payments received
Widget payCollectionTable(List<Map<String, dynamic>>
loans) {
    const boldStyle = TextStyle(fontWeight:
FontWeight.bold);

    return Expanded(
        child: Container(
            padding: EdgeInsets.all(16),
            decoration: BoxDecoration(
                color: Colors.white,
                borderRadius: BorderRadius.circular(8),
                boxShadow: [

```

```
BoxShadow(  
    color: Colors.black.withOpacity(0.05),  
    blurRadius: 8,  
    offset: Offset(0, 2),  
,  
],  
,  
child: LayoutBuilder(  
    builder: (context, constraints) {  
        return SingleChildScrollView(  
            scrollDirection: Axis.vertical,  
            child: SingleChildScrollView(  
                scrollDirection: Axis.horizontal,  
                child: ConstrainedBox(  
                    constraints: BoxConstraints(minWidth:  
constraints.maxWidth),  
                    child: DataTable(  
                        sortColumnIndex: sortColumnIndex,  
                        sortAscending: isAscending,  
                        columnSpacing: 58,  
                        columns: [  
                            DataColumn(label: Text("Payment ID", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans, "payID")),  
                            DataColumn(label: Text("Member Name",  
style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"memName")),  
                            DataColumn(label: Text("Member ID", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"memID")),  
                            DataColumn(label: Text("Loan ID", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"loanID")),  
                            DataColumn(label: Text("Payment Date",  
style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"payDate")),  
                            DataColumn(label: Text("Payment Method",  
style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"payMethod")),  
                            DataColumn(label: Text("Amount Paid", style:  
boldStyle), numeric: true, onSort: (i, asc) => onSort(i, asc,
```

```
loans, "amountPaid"))),
        DataColumn(label: Text("Collected By", style:
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,
"collectedBy")),
    ],
rows: loans.map((loan) {
    return DataRow(cells: [
        DataCell(Text(loan["payID"])),
        DataCell(Text(loan["memName"])),
        DataCell(Text(loan["memID"])),
        DataCell(Text(loan["loanID"])),
        DataCell(Text(loan["payDate"])),
        DataCell(Text(loan["payMethod"])),
        DataCell(Text("P${loan["amountPaid"]}")),
        DataCell(Text(loan["collectedBy"])),
    ]);
}),
),
),
),
);
},
),
),
),
);
}
}

// Show scheduled payments that were not made on
time.

Widget missedPayTable(List<Map<String, dynamic>>
loans) {
    const boldStyle = TextStyle(fontWeight:
FontWeight.bold);

    return Expanded(
        child: Container(
            padding: EdgeInsets.all(16),
            decoration: BoxDecoration(
                color: Colors.white,
                borderRadius: BorderRadius.circular(8),

```

```
boxShadow: [
    BoxShadow(
        color: Colors.black.withOpacity(0.05),
        blurRadius: 8,
        offset: Offset(0, 2),
    ),
],
),
),
child: LayoutBuilder(
    builder: (context, constraints) {
        return SingleChildScrollView(
            scrollDirection: Axis.vertical,
            child: SingleChildScrollView(
                scrollDirection: Axis.horizontal,
                child: ConstrainedBox(
                    constraints: BoxConstraints(minWidth:
constraints.maxWidth),
                    child: DataTable(
                        sortColumnIndex: sortColumnIndex,
                        sortAscending: isAscending,
                        columnSpacing: 58,
                        columns: [
                            DataColumn(label: Text("Loan ID", style:
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,
"loanID")),
                            DataColumn(label: Text("Member Name",
style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,
"memName")),
                            DataColumn(label: Text("Due Date", style:
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,
"dueDate")),
                            DataColumn(label: Text("Amount Due", style:
boldStyle), numeric: true, onSort: (i, asc) => onSort(i, asc,
loans, "amountDue")),
                            DataColumn(label: Text("Days Missed", style:
boldStyle), numeric: true, onSort: (i, asc) => onSort(i, asc,
loans, "daysMissed")),
                            DataColumn(label: Text("Contact No.", style:
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,
"contactNo")),
                        ],
                    ),
                ),
            ),
        ),
    ),
);
```

```
        DataColumn(label: Text("Next Pay Date",  
style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"nextPayDate")),  
    ],  
    rows: loans.map((loan) {  
        return DataRow(cells: [  
            DataCell(Text(loan["loanID"])),  
            DataCell(Text(loan["memName"])),  
            DataCell(Text(loan["dueDate"])),  
            DataCell(Text("P${loan["amountDue"]}")),  
            DataCell(Text(loan["daysMissed"])),  
            DataCell(Text(loan["contactNo"])),  
            DataCell(Text(loan["nextPayDate"])),  
        ]);  
    }).toList(),  
),  
(  
),  
);  
},  
),  
),  
);  
});  
}  
}
```

// Show income from fees, interest, and voucher-based transactions.

```
Widget voucherRevenueTable(List<Map<String,  
dynamic>> loans) {  
    const boldStyle = TextStyle(fontWeight:  
FontWeight.bold);
```

```
    return Expanded(  
        child: Container(  
            padding: EdgeInsets.all(16),  
            decoration: BoxDecoration(  
                color: Colors.white,  
                borderRadius: BorderRadius.circular(8),  
                boxShadow: [  
                    BoxShadow(  

```

```
color: Colors.black.withOpacity(0.05),  
blurRadius: 8,  
offset: Offset(0, 2),  
,  
],  
,  
child: LayoutBuilder(  
builder: (context, constraints) {  
return SingleChildScrollView(  
scrollDirection: Axis.vertical,  
child: SingleChildScrollView(  
scrollDirection: Axis.horizontal,  
child: ConstrainedBox(  
constraints: BoxConstraints(minWidth:  
constraints.maxWidth),  
child: DataTable(  
sortColumnIndex: sortColumnIndex,  
sortAscending: isAscending,  
columnSpacing: 58,  
columns: [  
 DataColumn(label: Text("Voucher ID", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"voucherID")),  
 DataColumn(label: Text("Date Issued", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"dateIssued")),  
 DataColumn(label: Text("Description", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans, "desc")),  
 DataColumn(label: Text("Amount Earned",  
style: boldStyle), numeric: true, onSort: (i, asc) => onSort(i,  
asc, loans, "amtEarned")),  
 DataColumn(label: Text("Revenue Type",  
style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"revType")),  
 DataColumn(label: Text("Recorded By", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"recordedBy")),  
],  
rows: loans.map((loan) {  
return DataRow(cells: [
```

```
        DataCell(Text(loan["voucherID"])),  
        DataCell(Text(loan["dateIssued"])),  
        DataCell(Text(loan["desc"])),  
        DataCell(Text("P${loan["amtEarned"]}")),  
        DataCell(Text(loan["revType"])),  
        DataCell(Text(loan["recordedBy"])),  
    ]);  
}).toList(),  
,  
,  
,  
);  
},  
),  
),  
);  
}  
}
```

```
@override  
Widget build(BuildContext context) {  
  return Scaffold(  
    backgroundColor: const Color(0xFFFFFFFF),  
    body: Column(  
      children: [  
  
        // top nav bar  
        const TopNavBar(splash: "Admin"),  
  
        // main area  
        Expanded(  
          child: Row(  
            crossAxisAlignment: CrossAxisAlignment.start,  
            children: [  
  
              // sidebar  
              const SideMenu(role: "Admin"),  
  
              // main content  
            ],  
          ),  
        ),  
      ],  
    ),  
  );  
}
```

```
Expanded(  
    child: Container(  
        margin: const EdgeInsets.only(left: 16),  
        padding: const  
            EdgeInsets.symmetric(horizontal: 24, vertical: 32),  
        child: ConstrainedBox(  
            constraints: const BoxConstraints(maxWidth:  
                900),  
            child: Column(  
                crossAxisAlignment:  
                    CrossAxisAlignment.stretch,  
                children: [  
  
                    // title  
                    const Text(  
                        "Reports Dashboard",  
                        style: TextStyle(fontSize: 28,  
                            fontWeight: FontWeight.bold),  
                    ),  
                    const Text(  
                        "Review various types of reports.",  
                        style: TextStyle(color: Colors.grey,  
                            fontSize: 14),  
                    ),  
                    SizedBox(height: 24),  
  
                    // buttons + filters row  
                    buttonsAndFiltersRow(),  
  
                    // table generated by filters  
                    Expanded(  
                        child: Container(  
                            margin: const EdgeInsets.only(top: 16),  
                            padding: const EdgeInsets.all(24),  
                            decoration: BoxDecoration(  
                                color: Colors.white,  
                                borderRadius:  
                                    BorderRadius.circular(12),  
                                boxShadow: [  
                                    BoxShadow(  

```



given the report type chosen, how do i make the different tables appear inside the main content

Excellent — your setup is already structured very cleanly 🎉 You just need to make the table **react to the selected report type**.

Here's exactly how you can make the correct table appear dynamically based on the dropdown selection 👇

---

### 🛠 Step 1 — Add state for the selected report type

At the top of your `_AdminRecordsState` class:

dart

 Copy code

```
String? selectedReportType;
```

---

### 🛠 Step 2 — Update the dropdown to store the selected value

In your `buttonsAndFiltersRow()` widget, modify the first dropdown:

dart

 Copy code

```
SizedBox(  
  width: 250,  
  child: DropdownInputField(  
    label: "Report Type",  
    items: [  
      "Active Loans",  
      "Overdue Loans",  
      "Member Loan Summary",  
      "Payment Collection",  
      "Missed Payments",  
      "Voucher & Revenue Summary"  
    ],  
    onChanged: (value) {  
      setState(() {  
        selectedReportType = value;  
      });  
    },  
  ),  
);
```

```
    selectedReportType = value;  
  }));  
},  
,  
,
```

👉 This assumes your `DropdownInputField` supports an `onChanged` callback.

If it doesn't yet, we can easily modify that widget to include it — I'll show how below.

---

### ✳️ Step 3 — Conditionally show the right table

Inside your `build() → Expanded → Container → Column → children`, find this block:

dart

 Copy code

```
Expanded(  
  child: Container(  
    margin: const EdgeInsets.only(top: 16),  
    padding: const EdgeInsets.all(24),  
    decoration: BoxDecoration(  
      color: Colors.white,  
      borderRadius: BorderRadius.circular(12),  
      boxShadow: [  
        BoxShadow(  
          color: Colors.black.withOpacity(0.1),  
          blurRadius: 6,  
          offset: const Offset(0, 4),  
        ),  
      ],  
    ),  
    child: Form(  
      child: SingleChildScrollView(  
        child: Column(  
          crossAxisAlignment: CrossAxisAlignment.start,  
          children: [  
            // main table  
          ]  
        ),  
      ),  
    ),
```

```
  ),  
  ),  
),
```

Replace the comment with:

dart

 Copy code

```
// Choose which table to display  
if (selectedReportType == "Active Loans")  
  activeLoansTable(activeLoansData)  
else if (selectedReportType == "Overdue Loans")  
  overdueLoansTable(overdueLoansData)  
else if (selectedReportType == "Member Loan Summary")  
  memberLoansTable(memberLoansData)  
else if (selectedReportType == "Payment Collection")  
  payCollectionTable(paymentCollectionData)  
else if (selectedReportType == "Missed Payments")  
  missedPayTable(missedPaymentsData)  
else if (selectedReportType == "Voucher & Revenue Summary")  
  voucherRevenueTable(voucherRevenueData)  
else  
  const Center(child: Text("Please select a report type to view.")),
```

 Replace `activeLoansData` , `overdueLoansData` , etc. with your actual data lists (can be fetched from Supabase or dummy data).

---

## Step 4 — (Optional) Update `DropdownInputField` to support `onChanged`

If your current `DropdownInputField` doesn't have `onChanged` , update it like this:

dart

 Copy code

```
class DropdownInputField extends StatelessWidget {  
  final String label;  
  final List<String> items;  
  final String? value;  
  final Function(String?)? onChanged;  
  final String? Function(String?)? validator;
```

```
const DropdownInputField({  
    super.key,  
    required this.label,  
    required this.items,  
    this.value,  
    this.onChanged,  
    this.validator,  
});  
  
@override  
Widget build(BuildContext context) {  
    return DropdownButtonFormField<String>(  
        decoration: InputDecoration(labelText: label),  
        value: value,  
        onChanged: onChanged,  
        validator: validator,  
        items: items.map((item) {  
            return DropdownMenuItem(  
                value: item,  
                child: Text(item),  
            );  
        }).toList(),  
    );  
}  
}
```

---

## Summary

You're basically doing:

- Add `selectedReportType` to state.
- Update when dropdown changes.
- Conditionally render the correct table based on that.

This is the standard, reactive Flutter pattern — and your existing `Expanded / Container` structure already makes it ideal for this.

---

Would you like me to show a quick working example using dummy data for all six report types?



this is my dropdowninputfield

```
class DropdownInputField extends StatelessWidget {  
    final String label;  
    final TextEditingController? controller;  
    final List<String> items;  
    final String? Function(String?)? validator;  
  
    const DropdownInputField({  
        super.key,  
        required this.label,  
        this.controller,  
        required this.items,  
        this.validator,  
    });  
  
    @override  
    Widget build(BuildContext context) {  
        return DropdownButtonFormField<String>(  
            value: controller?.text.isNotEmpty == true ?  
                controller!.text : null,  
            decoration: InputDecoration(labelText: label),  
            items: items  
                .map((item) => DropdownMenuItem(  
                    value: item,  
                    child: Text(item),  
                ))  
                .toList(),  
            onChanged: (value) {  
                if (controller != null) {  
                    controller!.text = value ?? "";  
                }  
            },  
        );  
    }  
}
```

```
validator: validator,  
);  
}  
}  
  
and what do i put inside the parameters here if i font have  
data yet  
// main table (sorting by time period not yet implemented)  
if (selectedReportType == "Active  
Loans")  
  
activeLoansTable(activeLoansData)  
else if (selectedReportType ==  
"Overdue Loans")  
  
overdueLoansTable(overdueLoansData)  
else if (selectedReportType ==  
"Member Loan Summary")  
  
memberLoansTable(memberLoansData)  
else if (selectedReportType ==  
"Payment Collection")  
  
payCollectionTable(paymentCollectionData)  
else if (selectedReportType ==  
"Missed Payments")  
  
missedPayTable(missedPaymentsData)  
else if (selectedReportType ==  
"Voucher & Revenue Summary")  
  
voucherRevenueTable(voucherRevenueData)  
else  
const Center(child: Text("Please  
select a report type to view.")),
```

Perfect — your `DropdownInputField` looks great 👍 It already stores the selection into the controller, so you don't even need to modify it to add `onChanged`.

Let's handle your second question now 

## If you don't have data yet, you can use dummy placeholders

For now, just pass in **empty lists** or mock data so your UI can render without crashing.

Example:

dart

 Copy code

```
// Temporary placeholder data
final List<Map<String, dynamic>> activeLoansData = [];
final List<Map<String, dynamic>> overdueLoansData = [];
final List<Map<String, dynamic>> memberLoansData = [];
final List<Map<String, dynamic>> paymentCollectionData = [];
final List<Map<String, dynamic>> missedPaymentsData = [];
final List<Map<String, dynamic>> voucherRevenueData = [];
```

You can declare those **above** your **build** method inside `_AdminRecordsState`.

Then your table calls like this:

dart

 Copy code

```
if (selectedReportType == "Active Loans")
    activeLoansTable(activeLoansData)
else if (selectedReportType == "Overdue Loans")
    overdueLoansTable(overdueLoansData)
else if (selectedReportType == "Member Loan Summary")
    memberLoansTable(memberLoansData)
else if (selectedReportType == "Payment Collection")
    payCollectionTable(paymentCollectionData)
else if (selectedReportType == "Missed Payments")
    missedPayTable(missedPaymentsData)
else if (selectedReportType == "Voucher & Revenue Summary")
    voucherRevenueTable(voucherRevenueData)
else
    const Center(child: Text("Please select a report type to view.")),
```

This way:

- You avoid null errors.
  - Your UI logic is already wired up.
  - You can easily replace the dummy lists later when your Supabase fetch or API call returns real data.
- 

### ✖ Optional: if you don't have the table functions yet

You can also temporarily stub those table widgets until you build them.

Example:

dart

Copy code

```
Widget activeLoansTable(List<Map<String, dynamic>> data) {  
    return const Center(child: Text("Active Loans Table (coming soon)"));  
}  
  
Widget overdueLoansTable(List<Map<String, dynamic>> data) {  
    return const Center(child: Text("Overdue Loans Table (coming soon)"));  
}  
  
// etc.
```

---

### ✓ TL;DR

- Just pass **empty lists** ( `[]` ) for now.
  - Create stub table widgets that return placeholder text.
  - Later, replace them with `DataTable` or custom widgets that use your fetched Supabase data.
- 

Would you like me to show sample **table column setups** for each report type next (e.g., what columns each should display once you have data)?



```
import 'package:flutter/material.dart';
import 'package:the_basics/widgets/top_navbar.dart';
import 'package:the_basics/widgets/side_menu.dart';
import 'package:the_basics/widgets/input_fields.dart';
import 'package:supabase_flutter/supabase_flutter.dart';

class AdminRecords extends StatefulWidget {
  const AdminRecords({super.key});

  @override
  State<AdminRecords> createState() =>
  _AdminRecordsState();
}

class _AdminRecordsState extends State<AdminRecords> {
  int? sortColumnIndex;
  bool isAscending = true;
  String? selectedReportType;

  // Temporary placeholder data
  final List<Map<String, dynamic>> activeLoansData = [];
  final List<Map<String, dynamic>> overdueLoansData =
  [];
  final List<Map<String, dynamic>> memberLoansData =
  [];
  final List<Map<String, dynamic>>
  paymentCollectionData = [];
  final List<Map<String, dynamic>> missedPaymentsData =
  [];
  final List<Map<String, dynamic>> voucherRevenueData =
  [];

  Widget buttonsAndFiltersRow() {
    return Row(
      children: [
        SizedBox(
          width: 250,
```

```
child: DropdownInputField(  
    label: "Report Type",  
    items: [  
        "Active Loans",  
        "Overdue Loans",  
        "Member Loan Summary",  
        "Payment Collection",  
        "Missed Payments",  
        "Voucher & Revenue Summary"  
    ],  
    onChanged: (value) {  
        setState(() {  
            selectedReportType = value;  
        });  
    },  
),  
),  
SizedBox(width: 16),  
  
SizedBox(  
    width: 250,  
    child: DropdownInputField(  
        label: "Filter by...",  
        items: [  
            "Month",  
            "Quarter",  
            "Year",  
        ],  
    ),  
),  
SizedBox(width: 16),  
  
SizedBox(  
    height: 28,  
    child: ElevatedButton.icon(  
        onPressed: () {},  
        label: const Text(  
            "Generate Report",  
            style: TextStyle(color: Colors.white),  
        ),
```

```
style: ElevatedButton.styleFrom(
    backgroundColor: Colors.black,
    minimumSize: const Size(100, 28),
    padding:
        const EdgeInsets.symmetric(horizontal: 8),
),
),
),
),
Spacer(),
SizedBox(
    height: 28,
    child: ElevatedButton.icon(
        onPressed: () {},
        icon: const Icon(Icons.download,
            color: Colors.white),
        label: const Text(
            "Download",
            style: TextStyle(color: Colors.white),
        ),
        style: ElevatedButton.styleFrom(
            backgroundColor: Colors.black,
            minimumSize: const Size(100, 28),
            padding:
                const EdgeInsets.symmetric(horizontal: 8),
),
),
),
),
],
);
}
}

void onSort<T>(
    int columnIndex,
    bool ascending,
    List<Map<String, dynamic>> data,
    String key,
) {
    setState(() {

```

```
sortColumnIndex = columnIndex;
isAscending = ascending;

data.sort((a, b) {
    final valueA = a[key];
    final valueB = b[key];

    if (valueA == null || valueB == null) return 0;

    if (valueA is num && valueB is num) {
        return ascending ? valueA.compareTo(valueB) :
valueB.compareTo(valueA);
    } else if (valueA is DateTime && valueB is DateTime) {
        return ascending ? valueA.compareTo(valueB) :
valueB.compareTo(valueA);
    } else {
        return ascending
            ? valueA.toString().compareTo(valueB.toString())
            : valueB.toString().compareTo(valueA.toString());
    }
});

});

});

}

// all loans currently being paid (not yet completed or
// overdue).
Widget activeLoansTable(List<Map<String, dynamic>>
loans) {
    const boldStyle = TextStyle(fontWeight:
FontWeight.bold);

    return Expanded(
        child: Container(
            padding: EdgeInsets.all(16),
            decoration: BoxDecoration(
                color: Colors.white,
                borderRadius: BorderRadius.circular(8),
                boxShadow: [
                    BoxShadow(
                        color: Colors.black.withOpacity(0.05),

```

```
        blurRadius: 8,  
        offset: Offset(0, 2),  
    ),  
],  
,  
child: LayoutBuilder(  
    builder: (context, constraints) {  
        return SingleChildScrollView(  
            scrollDirection: Axis.vertical,  
            child: SingleChildScrollView(  
                scrollDirection: Axis.horizontal,  
                child: ConstrainedBox(  
                    constraints: BoxConstraints(minWidth:  
constraints.maxWidth),  
                    child: DataTable(  
                        sortColumnIndex: sortColumnIndex,  
                        sortAscending: isAscending,  
                        columnSpacing: 58,  
                        columns: [  
                            DataColumn(label: Text("Loan ID", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"loanID")),  
                            DataColumn(label: Text("Member Name",  
style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"memName")),  
                            DataColumn(label: Text("Loan Type", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"loanType")),  
                            DataColumn(label: Text("Principal Amount",  
style: boldStyle), numeric: true, onSort: (i, asc) => onSort(i,  
asc, loans, "principalAmt")),  
                            DataColumn(label: Text("Remaining Balance",  
style: boldStyle), numeric: true, onSort: (i, asc) => onSort(i,  
asc, loans, "remainBal")),  
                            DataColumn(label: Text("Start Date", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"startDate")),  
                            DataColumn(label: Text("Due Date", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"dueDate")),
```

```
],
rows: loans.map((loan) {
    return DataRow(cells: [
        DataCell(Text(loan["loanID"].toString())),
        DataCell(Text(loan["memName"])),
        DataCell(Text(loan["loanType"])),
        DataCell(Text("P${loan["principalAmt"]}")),
        DataCell(Text("P${loan["remainBal"]}")),
        DataCell(Text(loan["startDate"])),
        DataCell(Text(loan["dueDate"])),
    ]);
}),
),
),
),
);
},
),
),
),
);
}
}

// loans with missed or late payments.
Widget overdueLoansTable(List<Map<String, dynamic>>
loans) {
    const boldStyle = TextStyle(fontWeight:
FontWeight.bold);

    return Expanded(
        child: Container(
            padding: EdgeInsets.all(16),
            decoration: BoxDecoration(
                color: Colors.white,
                borderRadius: BorderRadius.circular(8),
                boxShadow: [
                    BoxShadow(
                        color: Colors.black.withOpacity(0.05),
                        blurRadius: 8,
                        offset: Offset(0, 2),
                    ),
                ],
            ),
        ),
    );
}
```

```
        ],
),
child: LayoutBuilder(
    builder: (context, constraints) {
        return SingleChildScrollView(
            scrollDirection: Axis.vertical,
            child: SingleChildScrollView(
                scrollDirection: Axis.horizontal,
                child: ConstrainedBox(
                    constraints: BoxConstraints(minWidth:
constraints.maxWidth),
                    child: DataTable(
                        sortColumnIndex: sortColumnIndex,
                        sortAscending: isAscending,
                        columnSpacing: 58,
                        columns: [
                            DataColumn(label: Text("Loan ID", style:
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,
"loanID")),
                            DataColumn(label: Text("Member Name",
style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,
"memName")),
                            DataColumn(label: Text("Loan Type", style:
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,
"loanType")),
                            DataColumn(label: Text("Due Date", style:
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,
"dueDate")),
                            DataColumn(label: Text("Days Overdue",
style: boldStyle), numeric: true, onSort: (i, asc) => onSort(i,
asc, loans, "daysOverdue")),
                            DataColumn(label: Text("Amount Due", style:
boldStyle), numeric: true, onSort: (i, asc) => onSort(i, asc,
loans, "amountDue")),
                            DataColumn(label: Text("Accumulated Late
Fees", style: boldStyle), numeric: true, onSort: (i, asc) =>
onSort(i, asc, loans, "lateFees")),
                            DataColumn(label: Text("Contact No.", style:
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,
"contactNo")),
                        ],
                    ),
                ),
            ),
        ),
    ),
);
```

```
],
rows: loans.map((loan) {
    return DataRow(cells: [
        DataCell(Text(loan["loanID"].toString())),
        DataCell(Text(loan["memName"])),
        DataCell(Text(loan["loanType"])),
        DataCell(Text(loan["dueDate"])),  
  
        DataCell(Text(loan["daysOverdue"].toString())),
        DataCell(Text("P\${loan["amountDue"]}")),
        DataCell(Text("P\${loan["lateFees"]}")),
        DataCell(Text(loan["contactNo"])),
    ]);
}),  
),
),
),
);
},
),
),
),
);
}
}  
  
// each member's total loan activity.
Widget memberLoansTable(List<Map<String, dynamic>>
loans) {
    const boldStyle = TextStyle(fontWeight:
FontWeight.bold);  
  
    return Expanded(
        child: Container(
            padding: EdgeInsets.all(16),
            decoration: BoxDecoration(
                color: Colors.white,
                borderRadius: BorderRadius.circular(8),
                boxShadow: [
                    BoxShadow(
                        color: Colors.black.withOpacity(0.05),
                        blurRadius: 8,  
                ]  
            ),
        ),
    );
}
```

```
        offset: Offset(0, 2),  
        ),  
        ],  
        ),  
        child: LayoutBuilder(  
            builder: (context, constraints) {  
                return SingleChildScrollView(  
                    scrollDirection: Axis.vertical,  
                    child: SingleChildScrollView(  
                        scrollDirection: Axis.horizontal,  
                        child: ConstrainedBox(  
                            constraints: BoxConstraints(minWidth:  
                                constraints.maxWidth),  
                            child: DataTable(  
                                sortColumnIndex: sortColumnIndex,  
                                sortAscending: isAscending,  
                                columnSpacing: 58,  
                                columns: [  
                                    DataColumn(label: Text("Member Name",  
                                        style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
                                        "memName")),  
                                    DataColumn(label: Text("Member ID", style:  
                                        boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
                                        "memID")),  
                                    DataColumn(label: Text("Total Loans", style:  
                                        boldStyle), numeric: true, onSort: (i, asc) => onSort(i, asc,  
                                        loans, "totalLoans")),  
                                    DataColumn(label: Text("Total Borrowed",  
                                        style: boldStyle), numeric: true, onSort: (i, asc) => onSort(i, asc,  
                                        loans, "totalBorrowed")),  
                                    DataColumn(label: Text("Total Paid", style:  
                                        boldStyle), numeric: true, onSort: (i, asc) => onSort(i, asc,  
                                        loans, "totalPaid")),  
                                    DataColumn(label: Text("Outstanding  
Balance", style: boldStyle), numeric: true, onSort: (i, asc) =>  
onSort(i, asc, loans, "outBal")),  
                                    DataColumn(label: Text("Last Payment Date",  
                                        style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
                                        "lastPaid")),  
                                    DataColumn(label: Text("Loan Status", style:
```

```
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,
"loanStatus")),
],
rows: loans.map((loan) {
return DataRow(cells: [
DataCell(Text(loan["memName"])),
DataCell(Text(loan["memID"].toString())),
DataCell(Text("P${loan["totalLoans"]}")),
DataCell(Text("P${loan["totalBorrowed"]}")),
DataCell(Text("P${loan["totalPaid"]}")),
DataCell(Text("P${loan["outBal"]}")),
DataCell(Text(loan["lastPaid"])),
DataCell(Text(loan["loanStatus"])),
]);
}),
),
),
),
);
},
),
),
),
);
}
}

// Track all payments received
Widget payCollectionTable(List<Map<String, dynamic>>
loans) {
const boldStyle = TextStyle(fontWeight:
FontWeight.bold);

return Expanded(
child: Container(
padding: EdgeInsets.all(16),
decoration: BoxDecoration(
color: Colors.white,
borderRadius: BorderRadius.circular(8),
boxShadow: [
BoxShadow(
color: Colors.black.withOpacity(0.05),

```

```
        blurRadius: 8,  
        offset: Offset(0, 2),  
    ),  
],  
,  
child: LayoutBuilder(  
    builder: (context, constraints) {  
        return SingleChildScrollView(  
            scrollDirection: Axis.vertical,  
            child: SingleChildScrollView(  
                scrollDirection: Axis.horizontal,  
                child: ConstrainedBox(  
                    constraints: BoxConstraints(minWidth:  
constraints.maxWidth),  
                    child: DataTable(  
                        sortColumnIndex: sortColumnIndex,  
                        sortAscending: isAscending,  
                        columnSpacing: 58,  
                        columns: [  
                            DataColumn(label: Text("Payment ID", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans, "payID")),  
                            DataColumn(label: Text("Member Name",  
style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"memName")),  
                            DataColumn(label: Text("Member ID", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"memID")),  
                            DataColumn(label: Text("Loan ID", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"loanID")),  
                            DataColumn(label: Text("Payment Date",  
style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"payDate")),  
                            DataColumn(label: Text("Payment Method",  
style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"payMethod")),  
                            DataColumn(label: Text("Amount Paid", style:  
boldStyle), numeric: true, onSort: (i, asc) => onSort(i, asc,  
loans, "amountPaid")),  
                            DataColumn(label: Text("Collected By", style:
```

```
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,
"collectedBy")),
],
rows: loans.map((loan) {
return DataRow(cells: [
DataCell(Text(loan["payID"])),
DataCell(Text(loan["memName"])),
DataCell(Text(loan["memID"])),
DataCell(Text(loan["loanID"])),
DataCell(Text(loan["payDate"])),
DataCell(Text(loan["payMethod"])),
DataCell(Text("P${loan["amountPaid"]}")),
DataCell(Text(loan["collectedBy"])),
]);
}),
),
),
),
);
},
),
),
),
);
}
}

// Show scheduled payments that were not made on
time.

Widget missedPayTable(List<Map<String, dynamic>>
loans) {
const boldStyle = TextStyle(fontWeight:
FontWeight.bold);

return Expanded(
child: Container(
padding: EdgeInsets.all(16),
decoration: BoxDecoration(
color: Colors.white,
borderRadius: BorderRadius.circular(8),
boxShadow: [
BoxShadow(

```

```
color: Colors.black.withOpacity(0.05),  
blurRadius: 8,  
offset: Offset(0, 2),  
,  
],  
,  
child: LayoutBuilder(  
builder: (context, constraints) {  
return SingleChildScrollView(  
scrollDirection: Axis.vertical,  
child: SingleChildScrollView(  
scrollDirection: Axis.horizontal,  
child: ConstrainedBox(  
constraints: BoxConstraints(minWidth:  
constraints.maxWidth),  
child: DataTable(  
sortColumnIndex: sortColumnIndex,  
sortAscending: isAscending,  
columnSpacing: 58,  
columns: [  
 DataColumn(label: Text("Loan ID", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"loanID")),  
 DataColumn(label: Text("Member Name",  
style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"memName")),  
 DataColumn(label: Text("Due Date", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"dueDate")),  
 DataColumn(label: Text("Amount Due", style:  
boldStyle), numeric: true, onSort: (i, asc) => onSort(i, asc,  
loans, "amountDue")),  
 DataColumn(label: Text("Days Missed", style:  
boldStyle), numeric: true, onSort: (i, asc) => onSort(i, asc,  
loans, "daysMissed")),  
 DataColumn(label: Text("Contact No.", style:  
boldStyle), onSort: (i, asc) => onSort(i, asc, loans,  
"contactNo")),  
 DataColumn(label: Text("Next Pay Date",  
style: boldStyle), onSort: (i, asc) => onSort(i, asc, loans,
```

```
"nextPayDate")),
    ],
    rows: loans.map((loan) {
      return DataRow(cells: [
        DataCell(Text(loan["loanID"])),
        DataCell(Text(loan["memName"])),
        DataCell(Text(loan["dueDate"])),
        DataCell(Text("P${loan["amountDue"]}")),
        DataCell(Text(loan["daysMissed"])),
        DataCell(Text(loan["contactNo"])),
        DataCell(Text(loan["nextPayDate"])),
      ]);
    }).toList(),
  ),
),
),
),
),
);
},
),
),
),
);
}
}

// Show income from fees, interest, and voucher-based
transactions.

Widget voucherRevenueTable(List<Map<String,
dynamic>> loans) {
  const boldStyle = TextStyle(fontWeight:
  FontWeight.bold);

  return Expanded(
    child: Container(
      padding: EdgeInsets.all(16),
      decoration: BoxDecoration(
        color: Colors.white,
        borderRadius: BorderRadius.circular(8),
        boxShadow: [
          BoxShadow(
            color: Colors.black.withOpacity(0.05),
            blurRadius: 8,
```



```
        DataCell(Text(loan["desc"])),  
        DataCell(Text("P${loan["amtEarned"]}")),  
        DataCell(Text(loan["revType"])),  
        DataCell(Text(loan["recordedBy"])),  
    ]);  
}).toList(),  
,  
,  
,  
);  
,  
,  
,  
);  
}  
}
```

```
@override  
Widget build(BuildContext context) {  
  return Scaffold(  
    backgroundColor: const Color(0xFFFFFFFF),  
    body: Column(  
      children: [  
  
        // top nav bar  
        const TopNavBar(splash: "Admin"),  
  
        // main area  
        Expanded(  
          child: Row(  
            mainAxisAlignment: MainAxisAlignment.start,  
            children: [  
  
              // sidebar  
              const SideMenu(role: "Admin"),  
  
              // main content  
              Expanded(  
                child: Container(  
                  color: Colors.white,
```

```
margin: const EdgeInsets.only(left: 16),
padding: const
EdgeInsets.symmetric(horizontal: 24, vertical: 32),
child: ConstrainedBox(
constraints: const BoxConstraints(maxWidth:
900),
child: Column(
crossAxisAlignment:
CrossAxisAlignment.stretch,
children: [
// title
const Text(
"Reports Dashboard",
style: TextStyle(fontSize: 28,
fontWeight: FontWeight.bold),
),
const Text(
"Review various types of reports.",
style: TextStyle(color: Colors.grey,
fontSize: 14),
),
SizedBox(height: 24),
// buttons + filters row
buttonsAndFiltersRow(),
// table generated by filters
Expanded(
child: Container(
margin: const EdgeInsets.only(top: 16),
padding: const EdgeInsets.all(24),
decoration: BoxDecoration(
color: Colors.white,
borderRadius:
BorderRadius.circular(12),
boxShadow: [
BoxShadow(
color: Colors.black.withOpacity(0.1),
blurRadius: 6,
```

```
        offset: const Offset(0, 4),
    ),
],
),
),

child: Form(
    child: SingleChildScrollView(
        child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
                // main table (sorting by time
                period not yet implemented)
                if (selectedReportType == "Active
                    Loans")
                    activeLoansTable(activeLoansData)
                else if (selectedReportType ===
                    "Overdue Loans")
                    overdueLoansTable(overdueLoansData)
                else if (selectedReportType ===
                    "Member Loan Summary")
                    memberLoansTable(memberLoansData)
                else if (selectedReportType ===
                    "Payment Collection")
                    payCollectionTable(paymentCollectionData)
                else if (selectedReportType ===
                    "Missed Payments")
                    missedPayTable(missedPaymentsData)
                else if (selectedReportType ===
                    "Voucher & Revenue Summary")
                    voucherRevenueTable(voucherRevenueData)
                else
                    const Center(child: Text("Please
```

select a report type to view.")),

```
        ],
        ),
        ),
        )

        ),
        )

        ],
        ),
        ),
        ),
        ),
        ),
        ),
        );
    }
}
```

```
class DropdownInputField extends StatelessWidget {
    final String label;
    final TextEditingController? controller;
    final List<String> items;
    final String? Function(String?)? validator;

    const DropdownInputField({
        super.key,
        required this.label,
        this.controller,
        required this.items,
        this.validator,
    });
}
```

```

@Override
Widget build(BuildContext context) {
    return DropdownButtonFormField<String>(
        value: controller?.text.isNotEmpty == true ?
        controller!.text : null,
        decoration: InputDecoration(labelText: label),
        items: items
            .map((item) => DropdownMenuItem(
                value: item,
                child: Text(item),
            ))
            .toList(),
        onChanged: (value) {
            if (controller != null) {
                controller!.text = value ?? "";
            }
        },
        validator: validator,
    );
}
}
}

```

what wrong with the onchanged

Nice — you've got a well-structured AdminRecords screen 🎉

The issue you're running into with onChanged is simple: your DropdownInputField widget **doesn't actually define an onChanged parameter**, but you're trying to pass one when you call it here:

dart

 Copy code

```

DropdownInputField(
    label: "Report Type",
    items: [...],
    onChanged: (value) { // ✗ Error: onChanged is not defined in the class
        setState(() {
            selectedReportType = value;
        });
    });

```

```
},  
,
```

## ✓ Fix

You just need to **add an `onChanged` parameter** to your `DropdownInputField` class and call it inside the widget's internal `onChanged`.

Here's the fixed version:

dart

 Copy code

```
class DropdownInputField extends StatelessWidget {  
    final String label;  
    final TextEditingController? controller;  
    final List<String> items;  
    final String? Function(String?)? validator;  
    final void Function(String?)? onChanged; // ✓ Add this  
  
    const DropdownInputField({  
        super.key,  
        required this.label,  
        this.controller,  
        required this.items,  
        this.validator,  
        this.onChanged, // ✓ Add this  
    });  
  
    @override  
    Widget build(BuildContext context) {  
        return DropdownButtonFormField<String>(  
            value: controller?.text.isNotEmpty == true ? controller!.text : null,  
            decoration: InputDecoration(labelText: label),  
            items: items  
            .map((item) => DropdownMenuItem(  
                value: item,  
                child: Text(item),  
            ))  
            .toList(),
```

```
        onChanged: (value) {
            // ✓ Update both controller and parent state callback
            if (controller != null) controller!.text = value ?? '';
            if (onChanged != null) onChanged!(value);
        },
        validator: validator,
    );
}
}
```

## ✓ Why this works

Now your dropdown:

- Updates the attached `TextEditingController` (if any)
- Triggers the parent widget's `setState()` via your callback
- Keeps both the parent (`selectedReportType`) and controller (if used) in sync

After this fix, your usage in `buttonsAndFiltersRow()` will work perfectly:

dart

Copy code

```
DropdownInputField(
    label: "Report Type",
    items: [
        "Active Loans",
        "Overdue Loans",
        "Member Loan Summary",
        "Payment Collection",
        "Missed Payments",
        "Voucher & Revenue Summary"
    ],
    onChanged: (value) {
        setState(() {
            selectedReportType = value;
        });
    });
}
```

```
},  
,
```

---

Would you like me to make it so the dropdown **remembers the selected value automatically** even without a controller? (that can simplify your code further).



yes please

Perfect 👍 — here's an **improved version** of your `DropdownInputField` that:

- keeps track of its own selected value automatically
  - still supports an external controller (optional)
  - notifies the parent when the selection changes
- 

## ✳️ Updated `DropdownInputField`

dart

Copy code

```
import 'package:flutter/material.dart';  
  
class DropdownInputField extends StatefulWidget {  
  final String label;  
  final TextEditingController? controller;  
  final List<String> items;  
  final String? Function(String?)? validator;  
  final void Function(String?)? onChanged;  
  
  const DropdownInputField({  
    super.key,  
    required this.label,  
    this.controller,  
    required this.items,  
    this.validator,  
  }) ;}
```

```
        this.onChanged,
    });

    @override
    State<DropdownInputField> createState() => _DropdownInputFieldState();
}

class _DropdownInputFieldState extends State<DropdownInputField> {
    String? _selectedValue;

    @override
    void initState() {
        super.initState();
        // initialize from controller (if available)
        _selectedValue = widget.controller?.text.isNotEmpty == true
            ? widget.controller!.text
            : null;
    }

    @override
    Widget build(BuildContext context) {
        return DropdownButtonFormField<String>(
            value: _selectedValue,
            decoration: InputDecoration(labelText: widget.label),
            items: widget.items
                .map((item) => DropdownMenuItem(
                    value: item,
                    child: Text(item),
                ))
                .toList(),
            onChanged: (value) {
                setState(() {
                    _selectedValue = value;
                });
                // update controller (if used)
                if (widget.controller != null) {
                    widget.controller!.text = value ?? '';
                }
            });
}
}
```

```
// notify parent
if (widget.onChanged != null) {
    widget.onChanged!(value);
}
},
validator: widget.validator,
);
}
}
```

---

### ✓ Now you can use it like this:

If you want to **manually track the selection in your parent widget** (like for showing different tables):

dart

 Copy code

```
DropdownInputField(
label: "Report Type",
items: [
    "Active Loans",
    "Overdue Loans",
    "Member Loan Summary",
    "Payment Collection",
    "Missed Payments",
    "Voucher & Revenue Summary"
],
onChanged: (value) {
    setState(() {
        selectedReportType = value;
    });
},
),
```

Or, if you prefer to use a **TextEditingController** instead of a variable:

dart

 Copy code

```
final reportTypeController = TextEditingController();

DropdownInputField(
  label: "Report Type",
  controller: reportTypeController,
  items: [...],
),
```

Then access it anywhere as:

dart

 Copy code

```
print(reportTypeController.text);
```

---

Would you like me to make it also **show a default hint like “Select Report Type”** when nothing is chosen yet?



errr i dont want to change the dropdowninputfield anymore. i want na everytime the dropdown changes, the table contents also change

Ah! Got it 😊 — you don't want to change your `DropdownInputField` itself, you just want the **table to update whenever the dropdown selection changes**.

The key is to **use the `onChanged` callback you already have in your `DropdownInputField`** and call `setState` to update `selectedReportType`. That way, Flutter rebuilds the widget tree and shows the correct table.

Here's how you should do it in your **buttonsAndFiltersRow**:

dart

 Copy code

```
SizedBox(  
    width: 250,  
    child: DropdownInputField(  
        label: "Report Type",  
        items: [  
            "Active Loans",  
            "Overdue Loans",  
            "Member Loan Summary",  
            "Payment Collection",  
            "Missed Payments",  
            "Voucher & Revenue Summary"  
        ],  
        onChanged: (value) {  
            // Make sure to update state when dropdown changes  
            setState(() {  
                selectedReportType = value;  
            });  
        },  
    ),  
),
```

Then, in your table section:

```
dart

// main table (sorting by time period not yet implemented)
if (selectedReportType == "Active Loans")
    activeLoansTable(activeLoansData)
else if (selectedReportType == "Overdue Loans")
    overdueLoansTable(overdueLoansData)
else if (selectedReportType == "Member Loan Summary")
    memberLoansTable(memberLoansData)
else if (selectedReportType == "Payment Collection")
    payCollectionTable(paymentCollectionData)
else if (selectedReportType == "Missed Payments")
    missedPayTable(missedPaymentsData)
else if (selectedReportType == "Voucher & Revenue Summary")
    voucherRevenueTable(voucherRevenueData)
```



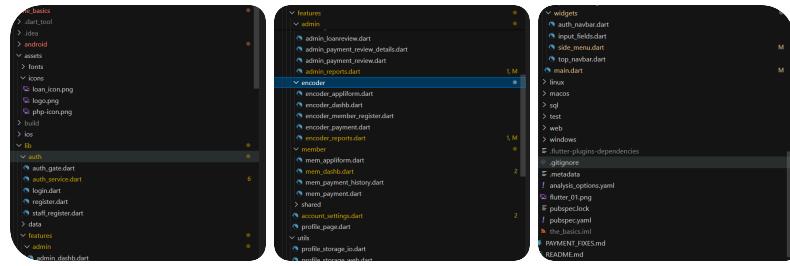
```
else
```

```
    const Center(child: Text("Please select a report type to view.")),
```

- That's it — now every time the user picks a different report type, `selectedReportType` updates, `setState` triggers a rebuild, and the correct table is shown.

If you want, I can also show a **quick way to make the table update instantly even if the data arrays are empty at first** — so you don't get errors with empty tables.

Do you want me to do that?



this is what my directory looks like at the moment. features include the pages meant for every role of our application. every page is a combination of ui and back end logic. data includes json for data to be displayed by these pages. utils contain things pertaining to profile storage. lastly, widgets contain widgets common to all pages, also containing ui and back end. i want to implement the right modularity. how do i organize it all

Perfect — your current structure already has a good foundation, but we can modularize it better by **separating UI, logic, and data layers** more clearly while keeping it intuitive for your team.

Here's a **recommended modular folder structure** for your app based on what you showed:

## Proposed Structure

bash

 Copy code

```
lib/
|
|   core/
|   |   constants/      # app-wide constants (colors, styles, etc.)
|   |   utils/          # general-purpose helpers, formatters, etc.
|   |   services/        # APIs, database, Supabase, authentication
|   |   widgets/         # shared widgets (buttons, input fields, navbar, etc.)
|
|   data/
|   |   models/          # Dart models / data classes (Loan, Member, etc.)
|   |   repositories/    # Data fetching logic, local & remote handling
|   |   mock/             # JSON / fake data for development
|
|   features/
|   |   admin/
|   |   |   presentation/ # UI (screens, widgets specific to admin)
|   |   |   logic/        # state mgmt (controllers, providers, etc.)
|   |   |   data/          # feature-specific data if needed
|
|   |   encoder/
|   |   |   presentation/
|   |   |   logic/
|   |   |   data/
|
|   |   member/
|   |   |   presentation/
|   |   |   logic/
|   |   |   data/
|
|   app/
|   |   router.dart       # handles navigation routes
|   |   app.dart           # wraps MaterialApp, theme, etc.
|
|   main.dart             # entry point
```

 Why This Works