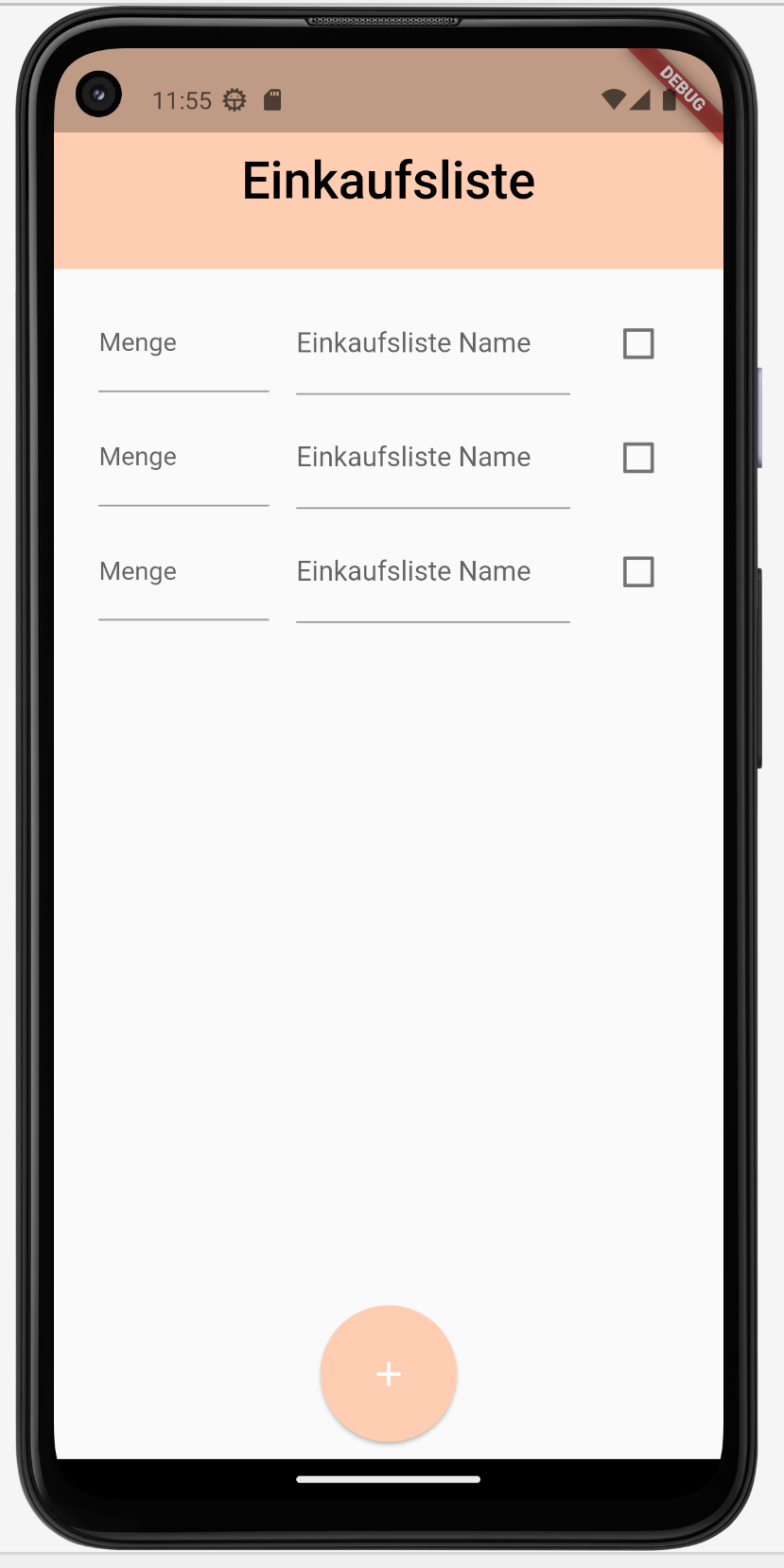
EinkaufslistenAPP



Klasse :

class ArtikelListe {  
 *// Eigenschaften*

String name;  
 int menge;  
 bool checked;  
  
 ArtikelListe(this.name, this.menge, this.checked); *// mit Konstruktor*}

import 'package:flutter/material.dart';  
import 'package:testeins/model/einkaufen.dart';  
import 'package:flutter\_slidable/flutter\_slidable.dart';  
  
class ToDoEinkaufen extends StatefulWidget {  
 const ToDoEinkaufen({Key? key}) : super(key: key);  
  
 @override  
 State<ToDoEinkaufen> createState() => \_ToDoEinkaufenState();  
}  
  
class \_ToDoEinkaufenState extends State<ToDoEinkaufen> {  
 List<ArtikelListe> liste = [];  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: PreferredSize(  
 preferredSize: Size(double.infinity, 80),  
 child: AppBar(  
 centerTitle: true,  
 title: Text(  
 'Einkaufsliste',  
 style: TextStyle(color: Colors.black, fontSize: 30),  
 ),  
 backgroundColor: Color(0xFFffcdb2),  
 elevation: 0,  
 ),  
 ),  
 body: Padding(  
 padding: const EdgeInsets.all(10.0),  
 child: Column(  
 children: [  
 Expanded(  
 child: ListView.builder(  
 itemCount: liste.length,  
 itemBuilder: (BuildContext context, int index) {  
 return Hinzufuegen(index);  
 },  
 ),  
 ),  
 SizedBox(height: 20),  
 Container(  
 width: 50,  
 height: 50,  
 child: ElevatedButton(  
 style: ButtonStyle(  
 alignment: Alignment.center,  
 shape: MaterialStateProperty.all(CircleBorder()),  
 backgroundColor: MaterialStateProperty.all(Color(0xFFffcdb2)),  
 ),  
 child: Icon(Icons.add),  
 onPressed: () {  
 setState(() {  
 liste.add(ArtikelListe("", 0,  
 false)); *// Füge ein Listenelement mit leerem Namen, Menge und nicht ausgewähltem Zustand hinzu* });  
 },  
 ),  
 ),  
 ],  
 ),  
 ),  
 );  
 }

**// zusätzliche Methoden:**

Slidable Hinzufuegen(int index) {  
 return Slidable(  
 actionPane: SlidableDrawerActionPane(),  
 secondaryActions: [  
 IconSlideAction(  
 caption: 'Löschen',  
 color: Colors.red,  
 icon: Icons.delete,  
 onTap: () {  
 setState(() {  
 liste.removeAt(index);  
 });  
 },  
 ),  
 ],  
 child: ListTile(  
 title: TextField(  
 *// in der Mitte* onChanged: (value) {  
 if (RegExp(r'^[a-zA-Z]+$').hasMatch(value.trim())) {  
 setState(() {  
 liste[index].name = value;  
 });  
 } else {  
 PopupName();  
 setState(() {  
 liste[index].name = "";  
 });  
 }  
 },  
 decoration: InputDecoration(  
 labelText: 'Einkaufsliste Name',  
 ),  
 ),  
 leading: Container(  
 *// am Ende* width: 100,  
 child: TextField(  
 style: TextStyle(fontSize: 20),  
 textAlign: TextAlign.center,  
 onChanged: (value) {  
 if (RegExp(r'^\d{1,2}$').hasMatch(value.toString())) {  
 setState(() {  
 liste[index].menge = int.parse(value);  
 });  
 } else {  
 PopupMenge();  
 }  
 },  
 decoration: InputDecoration(  
 labelStyle: TextStyle(fontSize: 15),  
 labelText: 'Menge',  
 ),  
 ),  
 ),  
 trailing: Checkbox(  
 *// am Anfang* value: liste[index].checked, *// Verwende den Wert aus der ArtikelListe* onChanged: (value) {  
 if (liste[index].name != null &&  
 liste[index].name.isNotEmpty &&  
 liste[index].menge != null &&  
 liste[index].menge > 0) {  
 *// Führen Sie den Code aus, wenn liste[index].name und liste[index].menge nicht leer oder null sind* setState(() {  
 liste[index].checked = value!;  
 });  
 } else {  
 PopupCheckbox();  
 }  
 },  
 ),  
 ),  
 );  
 }  
  
 void PopupMenge() {  
 showDialog(  
 context: context,  
 builder: (context) => AlertDialog(  
 title: Text("Der eingegebene Wert ist keine Zahl. Bitte korrigieren."),  
 actions: [  
 ElevatedButton(  
 child: Text("close"),  
 onPressed: () {  
 Navigator.of(context).pop();  
 setState(() {  
 *// liste[index].menge = // Das Mengenfeld leeren* });  
 },  
 ),  
 ],  
 ),  
 );  
 }  
  
 void PopupName() {  
 showDialog(  
 context: context,  
 builder: (context) => AlertDialog(  
 title: Text(  
 "Bitte einen Namen des Einkaufsartikels ein. Bitte korrigieren."),  
 actions: [  
 ElevatedButton(  
 child: Text("close"),  
 onPressed: () {  
 Navigator.of(context).pop();  
 },  
 ),  
 ],  
 ),  
 );  
 }  
  
 void PopupCheckbox() {  
 showDialog(  
 context: context,  
 builder: (context) => AlertDialog(  
 title: Text(  
 "Bitte gib eine Menge und den Namen des Einkaufsartikels ein."),  
 actions: [  
 ElevatedButton(  
 child: Text("close"),  
 onPressed: () {  
 Navigator.of(context).pop();  
 },  
 ),  
 ],  
 ),  
 );  
 }  
}

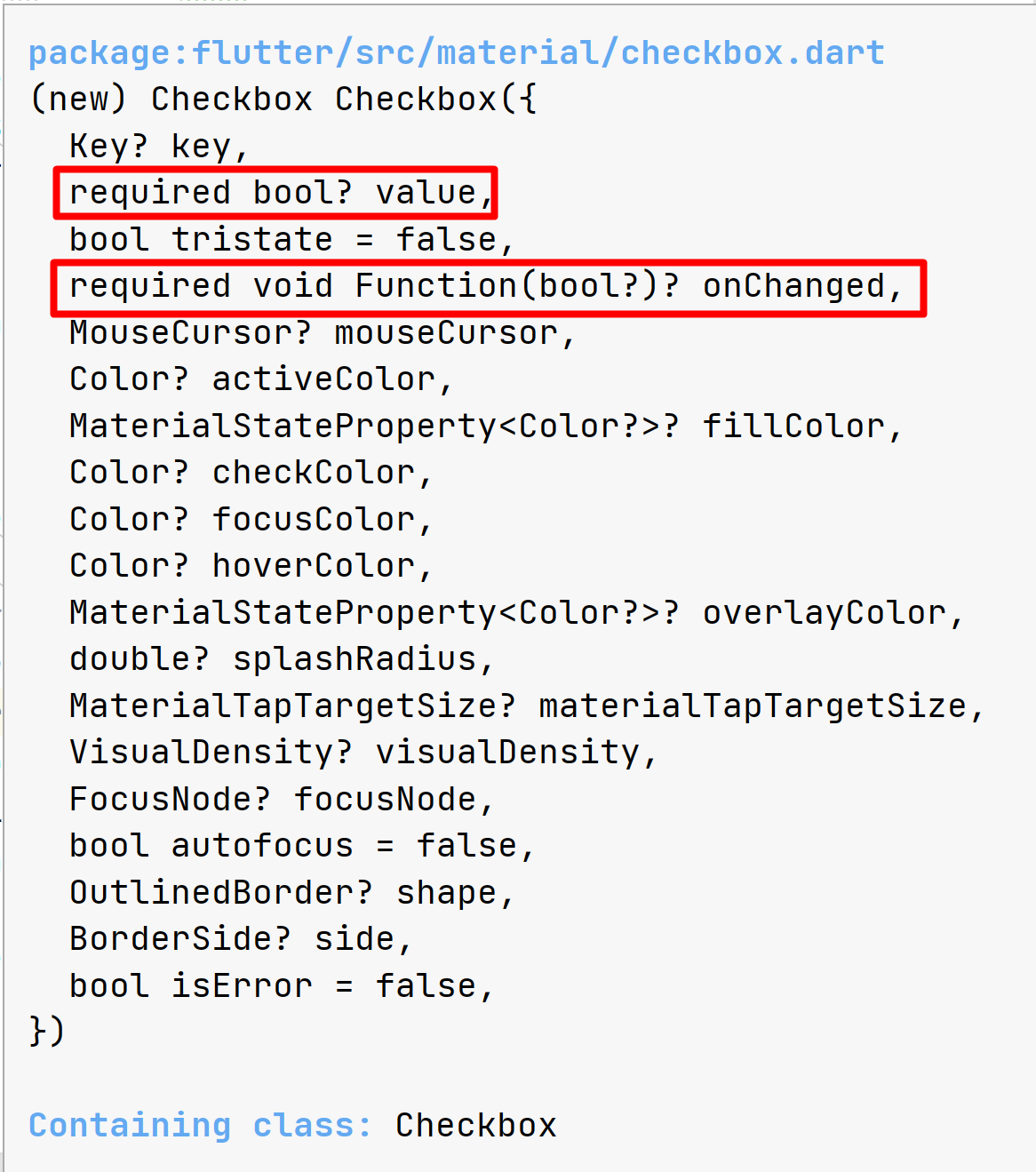
Erklärungen

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

trailing: Checkbox( *// am Anfang* value: liste[index].checked,

*// Verwende den Wert aus der ArtikelListe, also false* onChanged: (value) {  
 setState(() {  
 liste[index].checked = value!;

*// Aktualisiere den Zustand der Checkbox in der ArtikelListe* });},),);



Sicher, hier ist eine schrittweise Erklärung des Codes:

1. leading: Checkbox(: Dieser Teil definiert eine Checkbox-Widget, die als führendes (also am Anfang steht) Element in der ListTile angezeigt wird.
2. value: liste[index].checked,: Hier wird der Wert der Checkbox festgelegt. Wir verwenden liste[index].checked, um den aktuellen Zustand der Checkbox aus der ArtikelListe zu erhalten. Das bedeutet, dass der angezeigte Zustand der Checkbox mit dem Wert von liste[index].checked übereinstimmt. => in unserem Fall : false
3. onChanged: (value) {: Dieser Teil definiert, was passieren soll, wenn der Benutzer den Zustand der Checkbox ändert. Eine Funktion wird aufgerufen und der geänderte Wert wird als Parameter value übergeben.
4. setState(() {: setState ist eine Methode, die verwendet wird, um den Zustand des Widgets zu aktualisieren und eine Aktualisierung der Benutzeroberfläche auszulösen. Hier wird setState aufgerufen, um anzugeben, dass sich der Zustand geändert hat.
5. liste[index].checked = value ?? false;: In diesem Schritt wird der Zustand der Checkbox in der ArtikelListe aktualisiert. liste[index].checked wird mit dem Wert von value aktualisiert, wobei value der geänderte Zustand der Checkbox ist. Der Null-Prüfungsoperator ?? wird verwendet, um den Standardwert false zu setzen, falls der Wert von value null ist.

Zusammenfassend aktualisiert dieser Code also den Zustand der Checkbox in der ArtikelListe, wenn der Benutzer den Zustand ändert, und sorgt dafür, dass die Checkbox den aktuellen Wert aus der Liste anzeigt.

## Onchanged Funktion:

1. (value) {: **Die Funktion onChanged wird aufgerufen, wenn der Benutzer den Zustand der Checkbox ändert.** Der Parameter value wird übergeben und enthält den geänderten Wert der Checkbox.
2. setState(() {: Mit setState geben wir an, dass sich der Zustand des Widgets ändert und eine Aktualisierung der Benutzeroberfläche erforderlich ist. Dies ermöglicht es Flutter, das Widget neu zu zeichnen und die Änderungen anzuzeigen.
3. liste[index].checked = value!;: In diesem Schritt aktualisieren wir den Zustand der Checkbox in der ArtikelListe. value enthält den geänderten Wert der Checkbox. Wir setzen liste[index].checked auf den Wert von value. Das Ausrufezeichen (!) wird verwendet, um dem Compiler mitzuteilen, dass der Wert von value nicht null ist und als bool behandelt werden kann.

Zusammenfassend ermöglicht uns der onChanged-Teil, den Zustand der Checkbox zu überwachen und bei Änderungen entsprechende Aktionen auszuführen. Durch das Aktualisieren des Zustands in der ArtikelListe können wir den aktualisierten Zustand speichern und für weitere Verarbeitung oder Anzeige verwenden.

onChanged ist eine **Callback-Funktion**. Eine Callback-Funktion ist eine Funktion, die als Argument an eine andere Funktion übergeben wird und später von dieser Funktion aufgerufen wird, um einen bestimmten Codeblock auszuführen.

In diesem Fall wird die onChanged-Funktion als Callback-Funktion an die Checkbox-Komponente übergeben. Wenn der Benutzer den Zustand der Checkbox ändert, ruft die Checkbox-Komponente diese onChanged-Funktion auf und übergibt den geänderten Wert der Checkbox als Argument. Die übergebene Callback-Funktion (onChanged) wird dann ausgeführt, um den geänderten Wert zu verarbeiten und entsprechende Aktionen auszuführen, wie zum Beispiel das Aktualisieren des Zustands oder das Auslösen einer Benutzeroberflächenaktualisierung.

# warum liste[index].checked = value ?? false; und nicht liste[index].checked = value ;

Die Zeile liste[index].checked = value ?? false; verwendet den Null-Coalescing-Operator ??, um den Wert der Variable value zu überprüfen.

Der Null-Coalescing-Operator ?? ermöglicht es, einen Standardwert festzulegen, der verwendet wird, wenn der Ausdruck auf der linken Seite des Operators null ist. In diesem Fall wird value als null übergeben, wenn die Checkbox nicht aktiviert ist. Wenn value null ist, wird der Wert auf der rechten Seite des Operators, in diesem Fall false, verwendet und der checked-Wert auf false gesetzt.

Dies stellt sicher, dass selbst wenn value null ist, der checked-Wert in der ArtikelListe auf false gesetzt wird, um den nicht ausgewählten Zustand der Checkbox darzustellen.

Wenn wir stattdessen einfach liste[index].checked = value; verwenden würden und value null ist, würde dies zu einer Fehlermeldung führen, da der Typ Null nicht direkt einem bool-Typ zugewiesen werden kann.

# CallbackFunktionen in Flutter:

In Flutter werden Callback-Funktionen häufig verwendet, um Änderungen oder Benutzerinteraktionen zu verarbeiten. Callbacks ermöglichen die Kommunikation und den Datenaustausch zwischen verschiedenen Komponenten in der Flutter-Anwendung.

Einige gängige Szenarien, in denen Callback-Funktionen verwendet werden, sind:

1. Benutzerinteraktion: Wenn der Benutzer beispielsweise auf eine Schaltfläche klickt oder eine Eingabe in ein Textfeld eingibt, wird eine entsprechende Callback-Funktion aufgerufen, um die Benutzeraktion zu verarbeiten.
2. Zustandsänderungen: Wenn sich der Zustand einer Komponente ändert, kann eine Callback-Funktion aufgerufen werden, um diese Änderung zu verarbeiten und gegebenenfalls andere Komponenten zu aktualisieren.
3. Datenrückgabe: Manchmal muss eine Komponente Daten an eine andere Komponente zurückgeben. Hierbei wird eine Callback-Funktion verwendet, um die Daten von der empfangenden Komponente entgegenzunehmen und entsprechend zu verarbeiten.

Callbacks sind eine flexible Möglichkeit, Informationen und Aktionen zwischen Komponenten auszutauschen und ermöglichen eine effektive Kommunikation innerhalb der Flutter-Anwendung.

Verbesserter Code:

import 'package:flutter/material.dart';  
import 'package:testeins/model/einkaufen.dart';  
import 'package:flutter\_slidable/flutter\_slidable.dart';  
  
class ShoppingToDo extends StatefulWidget {  
 const ShoppingToDo({Key? key}) : super(key: key);  
  
 @override  
 State<ShoppingToDo> createState() => \_ShoppingToDoState();  
}  
  
class \_ShoppingToDoState extends State<ShoppingToDo> {  
 List<ShoppingItem> liste = [];  
 List<TextEditingController> nameControllers = [];  
 List<TextEditingController> amountControllers = [];

**// Dadurch, dass wir eine Liste kreieren, benötigen wir auch Controllerlisten, für jedes Feld ist ein Controller notwendig! Da wir zwei Typen von TextFields haben, benötigen wir auch 2 ControllerListen!**

@override  
 void dispose() {  
 for (var controller in nameControllers) {  
 controller.dispose();  
 }  
 for (var controller in amountControllers) {  
 controller.dispose();  
 }  
 super.dispose();  
 }

@override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: PreferredSize(  
 preferredSize: Size(double.infinity, 80),  
 child: AppBar(  
 centerTitle: true,  
 title: const Text(  
 'Einkaufsliste',  
 style: TextStyle(color: Colors.black, fontSize: 30),  
 ),  
 backgroundColor: Color(0xFFffcdb2),  
 elevation: 0,  
 ),  
 ),  
 body: Padding(  
 padding: EdgeInsets.all(10.0),  
 child: Column(  
 children: [  
 Expanded(  
 child: ListView.builder(  
 itemCount: liste.length,  
 itemBuilder: (BuildContext context, int index) {  
 if (index >= nameControllers.length) {  
 nameControllers.add(TextEditingController());  
 }  
 if (index >= amountControllers.length) {  
 amountControllers.add(TextEditingController());  
 }  
 return Hinzufuegen(index);  
 },  
 ),  
 ),  
 const SizedBox(height: 20),  
 Container(  
 width: 50,  
 height: 50,  
 child: ElevatedButton(  
 style: ButtonStyle(  
 alignment: Alignment.center,  
 shape: MaterialStateProperty.all(const CircleBorder()),  
 backgroundColor:  
 MaterialStateProperty.all(const Color(0xFFffcdb2)),  
 ),  
 child: Icon(Icons.add),  
 onPressed: () {  
 setState(() {  
 liste.add(ShoppingItem("", 0, false));  
 });  
 },  
 ),  
 ),  
 ],  
 ),  
 ),  
 );  
 }  
  
 Padding Hinzufuegen(int index) {  
 *// return Slidable(  
 // actionPane: SlidableDrawerActionPane(),  
 // secondaryActions: [  
 // IconSlideAction(  
 // caption: 'Löschen',  
 // color: Colors.red,  
 // icon: Icons.delete,  
 // onTap: () {  
 // setState(() {  
 // liste.removeAt(index);  
 // nameControllers.removeAt(index);  
 // amountControllers.removeAt(index);  
 // });  
 // },  
 // ),  
 // ],* return Padding(  
 padding: const EdgeInsets.all(2.0),  
 child: Row(  
 children: [  
 Expanded(  
 child: Container(  
 height: 70,  
 decoration: BoxDecoration(  
 *// borderRadius: BorderRadius.circular(10),* color: Colors.grey.shade200,  
 ),  
 child: ListTile(  
 minLeadingWidth:30,  
 title: TextField(  
 style: TextStyle(  
 decoration: liste[index].isChecked  
 ? TextDecoration.lineThrough  
 : TextDecoration.none,  
  
 ),  
  
 controller: nameControllers[index],  
 onChanged: (value) {  
 if (RegExp(r'^[a-zA-Z]+$').hasMatch(value.trim())) {  
 setState(() {  
 liste[index].name = value;  
 });  
 } else {  
 PopupName();  
 nameControllers[index].clear();  
 }  
 },  
 decoration: InputDecoration(  
 labelText: 'Item',  
  
 ),  
 ),  
 leading: Container(  
 width: 50,  
 child: TextField(  
 controller: amountControllers[index],  
 style: TextStyle(fontSize: 20),  
 textAlign: TextAlign.center,  
 onChanged: (value) {  
 if (RegExp(r'^\d{1,2}$').hasMatch(value.toString())) {  
 setState(() {  
 liste[index].amount = int.parse(value);  
 });  
 } else {  
 PopupMenge();  
 amountControllers[index].clear();  
 }  
 },  
 decoration: InputDecoration(  
 labelStyle: TextStyle(fontSize: 15),  
 labelText: 'Amount',  
 ),  
 ),  
 ),  
 trailing: Container(  
 width: 20,  
 child: Checkbox(  
 value: liste[index].isChecked,  
 onChanged: (value) {  
 if (liste[index].name.isNotEmpty && liste[index].amount > 0) {  
 setState(() {  
 liste[index].isChecked = value!;  
 });  
 } else {  
 PopupCheckbox();  
 }  
 },  
 ),  
 ),  
 ),  
 ),  
 ),  
 Container( *// delete Part* width: 50,  
 height: 70,  
 decoration: BoxDecoration(  
 *// borderRadius: BorderRadius.circular(10),* color: Colors.grey.shade200),  
 child: ElevatedButton(  
 style: ButtonStyle(  
 backgroundColor: MaterialStateProperty.all( Colors.grey),  
 ),  
 child: Icon(Icons.delete, color: Colors.red.shade100, size:25.0),  
 onPressed: (){  
 setState(() {  
 liste.removeAt(index);  
 nameControllers.removeAt(index);  
 amountControllers.removeAt(index);  
 });  
 },  
 ),  
 ),  
 ],  
 ),  
 );  
 }  
  
 void PopupMenge() {  
 showDialog(  
 context: context,  
 builder: (context) => AlertDialog(  
 title: Text("Der eingegebene Wert ist keine Zahl. Bitte korrigieren."),  
 actions: [  
 ElevatedButton(  
 child: Text("close"),  
 onPressed: () {  
 Navigator.of(context).pop();  
 },  
 ),  
 ],  
 ),  
 );  
 }  
  
 void PopupName() {  
 showDialog(  
 context: context,  
 builder: (context) => AlertDialog(  
 title: Text(  
 "Bitte einen Namen des Einkaufsartikels ein. Bitte korrigieren."),  
 actions: [  
 ElevatedButton(  
 child: Text("close"),  
 onPressed: () {  
 Navigator.of(context).pop();  
 },  
 ),  
 ],  
 ),  
 );  
 }  
  
 void PopupCheckbox() {  
 showDialog(  
 context: context,  
 builder: (context) => AlertDialog(  
 title: Text(  
 "Bitte gib eine Menge und den Namen des Einkaufsartikels ein."),  
 actions: [  
 ElevatedButton(  
 child: Text("close"),  
 onPressed: () {  
 Navigator.of(context).pop();  
 },  
 ),  
 ],  
 ),  
 );  
 }  
}  
*//*