Rīgas 64. vidusskola

**Naudas pārvaldības palīgprogramma**

Programmatūras prasību specifikācija

**Darba Autors:** Roberts Livmanis

Rīga 2023

**Saturs**

**1. Ievads** 3

1.1. Nolūks3

1.2. Programmas mērķis3

1.3. Pārskats3

**2. Vispārējais apraksts** 3

2.1. Produkta perspektīva3

2.2. Produkta funkcionalitāte3

2.3. Produkta lietotāji3

2.4. Vispārējie ierobežojumi4

**3. Programmatūras prasību specifikācija** 4

3.1. Funkcionālās prasības4

3.1.1. Kategoriju lapa4

3.1.1.1. Jaunu kategoriju veidošana4

3.1.1.2. Palīgfunkcijas5

3.1.1. Maksājumu lapa6

3.1.1.1. Jaunu maksājumu veidošana6

3.1.1.2. Palīgfunkcijas7

3.1.1. Galvenā lapa8

3.1.1.1. Datu attēlošana galvenajā lapā8

3.1.1.2. Palīgfunkcijas8

3.2. Nefunkcionālās prasības9

**4. Atkļūdošanas un akcepttestēšanas pārskats** 9

4.1. Vienību testēšanas pārskats9

4.2. Akcepttestēšanas pārskats10

4.2.1. Funkcionālo prasību izpildījums10

4.2.1. Nefunkcionālo prasību izpildījums10

**5. Lietotāja ceļvedis** 11

**6. Piemērotas licences pamatojums** 11

**Pielikumi** 12

**1. Ievads**

**1.1. Nolūks**

Dokuments ir sagatavots programmatūru prasību specifikāciju un skolas programmēšanas projektu darba ietvaros. Dokuments sevī ietver programmas plānošanu, tās izveides pārbaudes procesus un rezultātus.

**1.2. Programmas mērķis**

Naudas pārvaldības palīglīdzeklis ir paredzēts, lai tas palīdz cilvēkiem sekot savam budžeta plānam un automātiski aprēķina atlikušo budžetu, kā arī aprēķina budžetu, kas paredzēts katrai dzīves jomai. Programma nodrošina informāciju par pieejamajiem finanšu resursiem, uzskaita ienākumus un palīdz optimizēt finanšu plānošanu. Programma var palīdzēt sekmēt ilgtermiņa finanšu mērķu sasniegšanu.

**1.3. Pārskats:**

Pirmajā daļā ir ievadinformācija un vispārējs apraksts, kas satur dokumenta nolūku, definīciju skaidrojumu un sistēmas darbības sfēru.

**2. Vispārējais apraksts**

**2.1. Produkta perspektīva**

Naudas pārvaldības programmatūra pēdējos gados ir piedzīvojusi strauju popularitātes pieaugumu, jo arvien vairāk cilvēku atzīst efektīvas budžeta veidošanas un finanšu plānošanas nozīmi. Ņemot vērā, ka pieaug globālais pieprasījums pēc tādiem rīkiem, kas palīdz personām uzlabot finanšu pārvaldības prasmes un iegūt lielāku kontroli pār savām finansēm, šī produkta perspektīvas šķiet ļoti daudzsološas. Paredzams, ka šādu produktu tirgus turpinās paplašināties un turpmākajos gados šajā jomā būs ievērojamas izaugsmes un attīstības iespējas.

**2.2. Produkta funkcionalitāte**

1. Lietotāja ievaddatu analizēšana

2. Datu vizuālā attēlošana

3. Finanšu un budžeta aprēķini

**2.3. Produkta lietotāji**

Personas, kuras vēlas uzlabot savas finanšu vadības prasmes un iegūt lielāku kontroli pār savām finansēm. Tas varētu ietvert cilvēkus, kuri vēlas nepārtērēt un izveidot lielāku budžetu, ietaupīt konkrētiem finanšu mērķiem, sekot saviem izdevumiem vai vienkārši iegūt labāku izpratni par savu vispārējo finansiālo stāvokli.

**2.4. Vispārējie ierobežojumi**

Lai palaistu programmu, ir nepieciešams viedtālrunis, kā arī piekļuve internetam šī produkta lejupielādes brīdī.

**3. Programmatūras prasību specifikācija**

**3.1. Funkcionālās prasības**

**3.1.1. Kategoriju lapa**

Kategoriju lapa ļauj lietotājam izveidot kategorijas ar savu budžetu un apskatīt detalizētu informāciju par katru iepriekš izveidoto kategoriju.

*attēls 3.1.1. Kategorijas laukums*

Nosaukums: \_\_

Budžets: \_\_

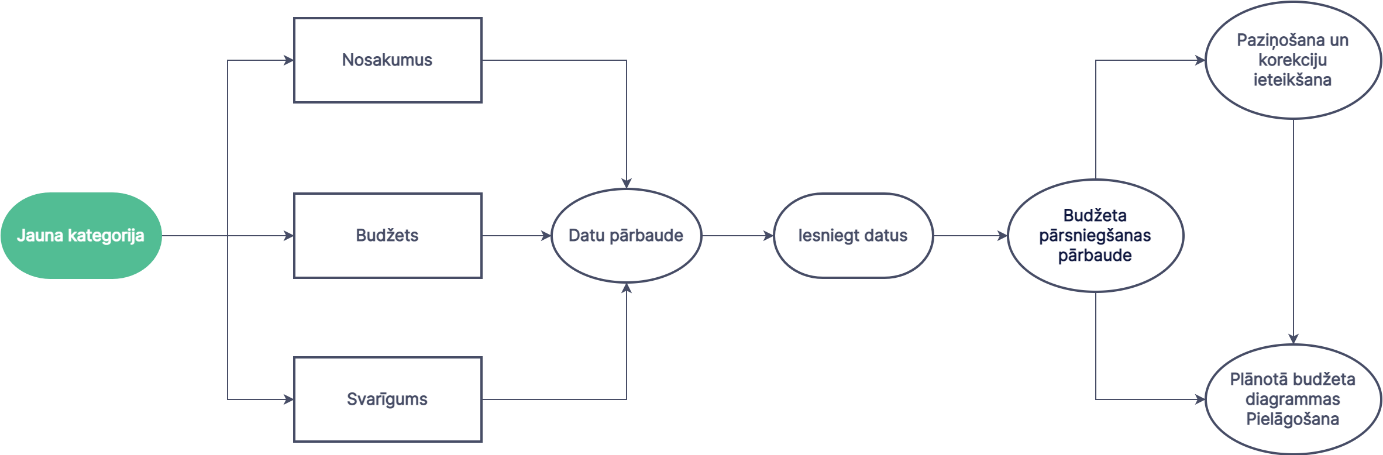
Svarīgums: \_\_

Atlikušais budžets:\_\_

Poga (Dzēst)

**3.1.1.1. Jaunu kategoriju veidošana**

*attēls 3.1.1.1. Kategoriju veidošanas modelis*

****

Pārsniedz

Nepārsniedz

Lietotājam ir iespēja izveidot jaunu kategoriju ar 3 parametriem:

1. Nosaukums

2. Budžets

3. Svarīgums (1-10).

Tiek pārbaudīta parametru atbilstība to datu tipiem un vai kategorijas plānotais budžets pārsniedz kopējo budžetu. Budžeta pārsniegšanas gadījumā programma informē lietotāju un samazina kategorijas budžetu tiktāl, lai netiktu pārsniegts kopējais budžets. Beigās tiek veiktas izmaiņas diagrammā un citu parametru attēlojumā galvenajā lapā.

Šī kategorija tiks parādīta kategoriju lapā un tās iespējams izveidot neierobežotā daudzumā. Ir arī iespēja rediģēt kategoriju un to dzēst.

**3.1.1.2. Palīgfunkcijas**

*tabula 3.1.1.2.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Funkcijas** | **Apraksts** | **Ievaddati** | **Izvaddati** |
| bool checkBudget(double) | Pārbauda, ​​vai kategorijas budžets pārsniedz kopējo budžetu | Kategoriju budžets | Pārsniedz vai nepārsniedz |
| double adjustBudget(double) | Aprēķina kategorijas budžetu, kas nepārsniedz kopējo budžetu | Kategoriju budžets | Jauns kategoriju budžets |
| List<Category> getCategories() | Vaicā datu bāzē kategoriju sarakstu un atgriež to. | Nav | Liste ar kategorijām |
| double calculateRemainingB(int, double) | Rediģējot kategorijas budžetu, funkcija aprēķina jauno kategorijas atlikušo budžetu, pamatojoties uz maksājumiem. | Kategoriju indekss (listē), jauns budžets | Jauns atlikušais budžets |
| deletePayments(String) | Dzēšot kategoriju, tiek dzēsti visi ar šo kategoriju saistītie maksājumi. | Kategoriju nosaukums | Nav |
| changePaymentCategoryNames(String, String) | Rediģējot kategorijas nosaukumu, funkcija maina maksājumu veco kategorijas nosaukumu uz jaunu. | Vecs kategorijas nosaukums, jauns kategorijas nosaukums | Nav |

Kategoriju lapas galvenā funkcija izmanto šīs palīgfunkcijas, lai veiksmīgi sekotu kategoriju lapas modelim un attēlotu informāciju (pilnu kodu sk. 1. pielikumā)

**3.1.2. Maksājumu lapa**

Maksājumu lapa ļauj lietotājam ierakstīt jaunus maksājumus ar esošas kategorijas nosaukumu, kā arī apskatīt detalizētu informāciju par katru iepriekš reģistrēto maksājumu.

*attēls 3.1.2. Maksājuma laukums*

Poga (Dzēst)

Kategorija: \_\_

Naudas summa:\_\_

Komentārs:\_\_

**3.1.2.1. Jaunu maksājumu veidošana**

*attēls 3.1.2.1. Maksājumu veidošanas modelis*

**Diagram

Description automatically generated**

Nepārsniedz

Pārsniedz

Nepietek

Pietek

Lietotājam ir iespēja pierakstīt jaunus maksājumus ar 3 parametriem:

1. Esošā kategorija (tiek dota izvēle no datubāzes ar esošām kategorijām)

2. Naudas summa

3. Komentārs (neobligāti)

Tiek pārbaudīta parametru atbilstība to datu tipiem un tiek pārbaudīts, vai kategorijas plānotais budžets tiek pārsniegts ar šo maksājumu. Budžeta pārsniegšanas gadījumā programma informē lietotāju un gadījumā, ja ir neizmantots budžets, programma palielina kategorijas budžetu un saglabā izmaiņas, ja nav neizmantota budžeta, tad, pamatojoties uz katru kategoriju svarīgumu un to pieejamo budžetu, tiek atņemts pieejamais budžets, lai “apmaksātu” summu. Beigās tiek veiktas izmaiņas diagrammas un citu parametru attēlojumā galvenajā lapā.

Šīs maksājums tiks parādīts maksājumu lapā, un tās iespējams izveidot neierobežotā daudzumā. Ir arī iespēja rediģēt maksājumu un to dzēst.

**3.1.2.2. Palīgfunkcijas**

*tabula 3.1.2.2.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Funkcijas** | **Apraksts** | **Ievaddati** | **Izvaddati** |
| bool checkCategoryB (double) | Pārbauda, ​​vai maksājums pārsniedz kategorijas budžetu | Maksājuma summa | Pārsniedz vai nepārsniedz |
| bool checkUnusedBudget (double) | Pārbauda, vai ir neizmantots budžets, ar kuru varētu palielināt kategorijas budžetu | Maksājuma summa | Ir vai nav |
| useUnusedBudget(double) | Izmantojot neizmantoto budžetu, palielina kategorijas budžetu | Maksājuma summa | Nav |
| decreaseCategoryBudget() | Meklē vismazāk svarīgāko kategoriju ar brīvo budžetu un atņem no tās budžeta daļu. | Maksājuma summa | Nav |
| List<Payment> getPayments() | Vaicā datu bāzē maksājumu sarakstu un atgriež to. | Nav | Liste ar maksājumiem |

Maksājumu lapas galvenā funkcija izmanto šīs palīgfunkcijas, lai veiksmīgi sekotu maksājuma lapas modelim un attēlotu informāciju (pilnu kodu sk. 3. pielikumā)

**3.1.3. Galvenā lapa**

Galvenā lapa ir paredzēta tikai informācijas apskatei, kas ir aprēķināta un savākta no visām pārējām lapām.

**3.1.3.1. Datu attēlošana galvenajā lapā**

Galvenā lapa apkopo datus no abām lapām, kategorijām un maksājumiem, kā arī aprēķina vispārīgu informāciju, kas tiek parādīta lietotāja ekrānā. Kopumā lapa lietotājam parādīs diagrammu ar kopējā budžeta sadalījumu kategorijās un četriem parametriem (sk. pielikumu):

1. Dienu skaits līdz perioda beigām (30 dienas)

2. Atlikušais budžets

3. Maksimālais budžets

4. Neizmantots budžets

**3.1.3.2. Palīgfunkcijas**

*tabula 3.1.3.2.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Funkcijas** | **Apraksts** | **Ievaddati** | **Izvaddati** |
| Map <String, double> getDataMap() | Pieprasa datubāzē visas esošas kategorijas un to plānoto budžetu un atgriež tos listē. | Nav | Liste ar visām kategorijām un to plānoto budžetu |
| double getRemainingBudget() | No maksimālā budžeta atņem visu maksājumu summu un rezultātu atgriež. | Nav | Atlikušais budžets |
| double getUnusedBudget() | No maksimālā budžeta atņem visu kategoriju plānoto budžetu un rezultātu atgriež. | Nav | Neizmantots budžets |
| double getMaxBudget() | Vaicā datu bāzē maksimālo budžetu un to atgriež. | Nav | Maksimālais budžets |
| int getDays() | Vaicā datu bāzē perioda sākuma datumu, aprēķina, cik dienas palika līdz perioda beigām un rezultātu atgriež. | Nav | Dienas līdz perioda beigām |

Galvenās lapas galvenā funkcija izmanto šīs palīgfunkcijas, lai attēlotu agrāk minēto informāciju (pilnu kodu sk. 5. pielikumā)

**3.1.4. Datubāze**

Programma izmanto "Hive" datu bāzi, kas ļauj saglabāt informāciju lokāli lietotāja ierīcē. Datubāze programmā tiek izmantota, lai saglabātu informāciju par visiem maksājumiem un esošajām kategorijām, kā arī 30 dienu perioda sākuma datumu un maksimālo budžetu (datubāzes relācijas modeli sk. 7. pielikumā)

**3.2. Nefunkcionālās prasības**

**3.2.1 Uzturamība**

Programmas kodam ir jābūt labi strukturētam, komentētam un viegli uztveramam,

jāievēro labs programmēšanas stils.

**3.2.2. Informācijas pieejamība un skaidrība**

Informācijai jābūt viegli saskatāmai un saprotamai. Lietotājs saprot, ko nozīmē katra ekrānā redzamā vērtība

**3.2.3. Lietojamība**

Programma ir viegli apgūstama un lietojama jauniem lietotājiem.

**4. Atkļūdošanas un akcepttestēšanas pārskats.**

**4.1. Vienību testēšanas pārskats**

*tabula 4.1.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Apraksts** | **Ievaddati** | **Sagaidāmais rezultāts** | **Reālais rezultāts** |
| Tiek pārbaudīta jaunas kategorijas izveide | Nosaukums ir tukšs | Programma neļauj izveidot kategoriju | Izpildās |
| Budžets ir negatīvs skaitlis | Programma neļauj izveidot kategoriju | Izpildās |
| Budžets nav skaitlis | Programma neļauj izveidot kategoriju | Izpildās |
| Svarīgums nav vesels skaitlis intervālā no 1 līdz 10 | Programma neļauj izveidot kategoriju | Izpildās |
| Budžets ir pozitīvs skaitlis, nosaukums nav tukšs un svarīgums ir skaitlis no 1 līdz 10 | Programma izveido kategoriju ar dotiem parametriem | Izpildās |

|  |  |  |  |
| --- | --- | --- | --- |
| **Apraksts** | **Ievaddati** | **Sagaidāmais rezultāts** | **Reālais rezultāts** |
| Tiek pārbaudīta jauna maksājuma izveide | Kategorija nav izvēlēta | Programma neļauj izveidot maksājumu | Izpildās |
| Summa ir negatīvs skaitlis | Programma neļauj izveidot maksājumu | Izpildās |
| Summa nav skaitlis | Programma neļauj izveidot maksājumu | Izpildās |
| Komentārs ir tukšs | Programma ļauj izveidot maksājumu | Izpildās |
| Kategorija ir izvēlēta, un summa ir pozitīvs skaitlis | Programma izveido maksājumu ar dotiem parametriem | Izpildās |

*tabula 4.1.1.*

**4.2. Akcepttestēšanas pārskats**

**4.2.1. Funkcionālo prasību izpildījums**

*tabula 4.2.1.*

|  |  |
| --- | --- |
| **Funkcionālā prasība** | **Izpildījums** |
| Korekta datu attēlošana uz kategoriju lapas (sk. 2. pielikumu) | Izpildās |
| Korekta datu attēlošana uz maksājumu lapas (sk. 4. pielikumu) | Izpildās |
| Korekta diagrammu un datu attēlošana uz galvenās lapas (sk. 6. pielikumu) | Izpildās |

**4.2.2. Funkcionālo prasību izpildījums**

*tabula 4.2.2.*

|  |  |
| --- | --- |
| **Nefunkcionālā prasība** | **Izpildījums** |
| Uzturamība | Izpildās |
| Informācijas pieejamība un skaidrība | Izpildās |
| Lietojamība | Izpildās |

**5. Lietotāja ceļvedis**

1. Pirmo reizi ieslēdzot programmu, jāievada budžets nākamajām 30 dienām.

2. Lapā ar kategorijām ir jāizveido savas kategorijas, kurām plānojat tērēt naudu, piemēram, pārtika, īre, atvaļinājums.

3. Pēc katra jauna pirkuma vai naudas tēriņiem ir jāieiet programmā un jāreģistrē jauns maksājums, norādot, kurai kategorijai šis maksājums pieder, un jānorāda maksājuma summa.

4. Programma aprēķinās izmaksas un parādīs lietotājam, vai un cik precīzi viņš ievēro plānoto budžetu.

**6. Piemērotas licences pamatojums**

Es izvēlējos licenci GNU GPL (GNU General Public License). GNU, jo tā garantē brīvību programmas izmantošanā un to var modificēt jebkādā veidā, ja vien šī mainītā versija tiek izplatīta kā bezmaksas programmatūra. Šī licence ļauj kodu izmantot ikvienam, vienlaikus nodrošinot arī modificēto versiju izlaišanu kā atvērtā pirmkoda produktus. Tas ietver pilnībā pārveidotā pirmkoda publiskošanu.

**Pielikumi**

**1. Pielikums**

import 'package:flutter/cupertino.dart';  
import 'package:flutter/material.dart';  
import 'package:flutter/services.dart';  
import 'package:hive\_flutter/adapters.dart';  
import '../Boxes.dart';  
import '../category.dart';  
import 'package:hive/hive.dart';  
  
class categoryPage extends StatefulWidget {  
 @override  
 \_CategoryPageState createState() => \_CategoryPageState();  
}  
  
class \_CategoryPageState extends State<categoryPage> {  
 final TextEditingController \_nameController = TextEditingController();  
 final TextEditingController \_budgetController = TextEditingController();  
 final TextEditingController \_importanceController = TextEditingController();  
  
 @override  
 Widget build(BuildContext context) {  
 List<Category> categories = Boxes.*getCategories*();  
 return Scaffold(  
 backgroundColor: CupertinoColors.*systemGroupedBackground*,  
 body: ListView.separated(  
 padding: EdgeInsets.all(20),  
 itemCount: categories.length,  
 itemBuilder: (BuildContext context, int index) {  
 final category = categories[index];  
 return ListTile(  
 contentPadding: EdgeInsets.all(20),  
 iconColor: Colors.*white*,  
 shape: RoundedRectangleBorder(  
 side: BorderSide(color: Colors.*white*, width: 2),  
 borderRadius: BorderRadius.circular(20)),  
 tileColor: Colors.*orange*,  
 title: Text(category!.name,  
 style: TextStyle(  
 color: Colors.*white*,  
 fontSize: 30,  
 fontWeight: FontWeight.*bold*)),  
 subtitle: Column(  
 crossAxisAlignment: CrossAxisAlignment.start,  
 children: [  
 Text('Budget: \$${category.budget.toStringAsFixed(2)}',  
 style: TextStyle(color: Colors.*white*, fontSize: 20)),  
 Text('Importance: ${category.importance}',  
 style: TextStyle(color: Colors.*white*, fontSize: 20)),  
 Text('Remaining: \$${category.reimaining.toStringAsFixed(2)}',  
 style: TextStyle(color: Colors.*white*, fontSize: 20)),  
 ],  
 ),  
 trailing: IconButton(  
 iconSize: 40,  
 icon: Icon(Icons.*delete*),  
 onPressed: () {  
 List<String> categoryNames = [];  
  
 for (int i = 0; i < categories.length; i++) {  
 Category cat = categories[0];  
 categoryNames.add(cat.name);  
 }  
 Category.*deletePayments*(categoryNames[index]);  
 setState(() {  
 Hive.box<Category>('CategoryBox').deleteAt(index);  
 });  
 },  
 ),  
 onTap: () {  
 \_nameController.text = category.name;  
 \_budgetController.text = category.budget.toString();  
 \_importanceController.text = category.importance.toString();  
 showDialog(  
 context: context,  
 builder: (BuildContext context) {  
 return AlertDialog(  
 title: Text('Edit Category'),  
 content: Column(  
 mainAxisSize: MainAxisSize.min,  
 children: [  
 TextField(  
 controller: \_nameController,  
 decoration: InputDecoration(  
 labelText: 'Name',  
 ),  
 ),  
 TextField(  
 controller: \_budgetController,  
 decoration: InputDecoration(  
 labelText: 'Budget',  
 ),  
 inputFormatters: <TextInputFormatter>[  
 FilteringTextInputFormatter.allow(  
 RegExp(r'^[1-9][0-9]\*')),  
 ],  
 keyboardType: TextInputType.numberWithOptions(  
 decimal: true,  
 ),  
 ),  
 TextField(  
 controller: \_importanceController,  
 decoration: InputDecoration(  
 labelText: 'Importance (1-10)',  
 ),  
 keyboardType: TextInputType.*number*,  
 inputFormatters: <TextInputFormatter>[  
 FilteringTextInputFormatter.allow(  
 RegExp("^(1[0-0]|[1-9])\$")),  
 ],  
 maxLength: 2,  
 maxLengthEnforcement: MaxLengthEnforcement.enforced,  
 ),  
 ],  
 ),  
 actions: [  
 TextButton(  
 child: Text('Cancel'),  
 onPressed: () {  
 Navigator.*of*(context).pop();  
 },  
 ),  
 TextButton(  
 child: Text('Save'),  
 onPressed: () {  
 Category.*changePaymentCategoryNames*(  
 category.name, \_nameController.text);  
 double newBudget =  
 double.*parse*(\_budgetController.text);  
 bool excess = Category.*checkBudget*(newBudget);  
 if (excess) {  
 newBudget = Category.*adjustBudget*(newBudget);  
 }  
 double remainingB = Category.*calculateRemainingB*(  
 index, double.*parse*(\_budgetController.text));  
 setState(() {  
 Hive.box<Category>('CategoryBox').putAt(  
 index,  
 Category(  
 \_nameController.text,  
 newBudget,  
 int.*parse*(\_importanceController.text),  
 remainingB));  
 });  
 Navigator.*of*(context).pop();  
 },  
 ),  
 ],  
 );  
 },  
 );  
 },  
 );  
 },

separatorBuilder: (context, index) => SizedBox(  
 height: 10,  
 )),  
 floatingActionButton: FloatingActionButton(  
 onPressed: () {  
 \_nameController.text = '';  
 \_budgetController.text = '';  
 \_importanceController.text = '';  
 showDialog(  
 context: context,  
 builder: (BuildContext context) {  
 return AlertDialog(  
 title: Text('New Category'),  
 content: Column(  
 mainAxisSize: MainAxisSize.min,  
 children: [  
 TextField(  
 controller: \_nameController,  
 decoration: InputDecoration(  
 labelText: 'Name',  
 ),  
 ),  
 TextField(  
 inputFormatters: <TextInputFormatter>[  
 FilteringTextInputFormatter.allow(  
 RegExp(r'^[1-9][0-9]\*')),  
 ],  
 controller: \_budgetController,  
 decoration: InputDecoration(  
 labelText: 'Budget',  
 )),  
 TextField(  
 controller: \_importanceController,  
 decoration: InputDecoration(  
 labelText: 'Importance (1-10)',  
 ),  
 keyboardType: TextInputType.*number*,  
 inputFormatters: <TextInputFormatter>[  
 FilteringTextInputFormatter.allow(  
 RegExp("^(1[0-0]|[1-9])\$")),  
 ],  
 maxLength: 2,  
 maxLengthEnforcement: MaxLengthEnforcement.enforced,  
 ),  
 ],  
 ),  
 actions: [  
 TextButton(  
 child: Text('Cancel'),  
 onPressed: () {  
 Navigator.*of*(context).pop();  
 },  
 ),  
 TextButton(  
 child: Text('Save'),  
 onPressed: () {  
 setState(() {  
 categories = Boxes.*getCategories*();  
 double newBudget = double.*parse*(\_budgetController.text);  
 bool excess = Category.*checkBudget*(newBudget);  
 if (excess) {  
 newBudget = Category.*adjustBudget*(newBudget);  
 }  
 Hive.box<Category>('CategoryBox').add(Category(  
 \_nameController.text,  
 newBudget,  
 int.*parse*(\_importanceController.text),  
 double.*parse*(\_budgetController.text),  
 ));  
 });  
 Navigator.*of*(context).pop();  
 },  
 ),  
 ],  
 );  
 },  
 );  
 },  
 backgroundColor: Colors.*orange*,  
 child: Icon(Icons.*add*),  
 ),  
 );  
 }  
}Diagram

Description automatically generated

**2. Pielikums**

**3. Pielikums**

import 'package:flutter/cupertino.dart';  
import 'package:flutter/material.dart';  
import 'package:flutter/services.dart';  
import 'package:hive/hive.dart';  
import '../Boxes.dart';  
import '../category.dart';  
  
class paymentPage extends StatefulWidget {  
 const paymentPage({Key? key}) : super(key: key);  
  
 @override  
 State<paymentPage> createState() => \_paymentPageState();  
}  
class \_paymentPageState extends State<paymentPage> {  
 final TextEditingController \_categoryController = TextEditingController();  
 final TextEditingController \_amountController = TextEditingController();  
 final TextEditingController \_commentController = TextEditingController();  
 String? value;  
   
 @override  
 Widget build(BuildContext context) {  
 List<Category> categories = Boxes.*getCategories*();  
 List<Payment> payments = Boxes.*getPayments*();  
 List<String> categoryNames = [];  
  
 for (int i = 0; i < categories.length; i++) {  
 Category cat = categories[0];  
 categoryNames.add(cat.name);  
 }  
 return Scaffold(  
 backgroundColor: CupertinoColors.*systemGroupedBackground*,  
 body: ListView.separated(  
 padding: EdgeInsets.all(20),  
 itemCount: payments.length,  
 itemBuilder: (BuildContext context, int index) {  
 final payment = payments[index];  
 *// String categoryName = categories[index].name;* return ListTile(  
 contentPadding: EdgeInsets.all(20),  
 iconColor: Colors.*white*,  
 shape: RoundedRectangleBorder(  
 side: BorderSide(color: Colors.*white*, width: 2),  
 borderRadius: BorderRadius.circular(20)),  
 tileColor: Colors.*orange*,  
 title: Text(payment!.categoryName,  
 style: TextStyle(  
 color: Colors.*white*,  
 fontSize: 30,  
 fontWeight: FontWeight.*bold*)),  
 subtitle: Column(  
 crossAxisAlignment: CrossAxisAlignment.start,  
 children: [  
 Text('Amount: \$${payment.amount.toStringAsFixed(2)}',  
 style: TextStyle(color: Colors.*white*, fontSize: 20)),  
 Text('Comment: ${payment.comment}',  
 style: TextStyle(color: Colors.*white*, fontSize: 20)),  
 ],  
 ),  
 trailing: IconButton(  
 iconSize: 40,  
 icon: Icon(Icons.*delete*),  
 onPressed: () {  
 setState(() {  
 Hive.box<Payment>('PaymentBox').deleteAt(index);  
 });  
 },  
 ),  
 onTap: () {  
 \_categoryController.text = payment.categoryName;  
 \_amountController.text = payment.amount.toString();  
 \_commentController.text = payment.comment;  
 showDialog(  
 context: context,  
 builder: (BuildContext context) {  
 return AlertDialog(  
 title: Text('Edit payment'),  
 content: Column(  
 mainAxisSize: MainAxisSize.min,  
 children: [  
 TextField(  
 controller: \_categoryController,  
 decoration: InputDecoration(  
 labelText: 'Category',  
 ),  
 ),  
 TextField(  
 inputFormatters: <TextInputFormatter>[  
 FilteringTextInputFormatter.allow(  
 RegExp(r'^[1-9][0-9]\*')),  
 ],  
 controller: \_amountController,  
 decoration: InputDecoration(  
 labelText: 'Amount',  
 ),  
 keyboardType: TextInputType.numberWithOptions(  
 decimal: true,  
 ),  
 ),  
 TextField(  
 controller: \_commentController,  
 decoration: InputDecoration(  
 labelText: 'Comment',  
 ),  
 ),  
 ],  
 ),  
 actions: [  
 TextButton(  
 child: Text('Cancel'),  
 onPressed: () {  
 Navigator.*of*(context).pop();  
 },  
 ),  
 TextButton(  
 child: Text('Save'),  
 onPressed: () {  
 double newAmount =  
 double.*parse*(\_amountController.text);  
 bool checkCatBudget = Category.*checkCategoryB*(newAmount);  
 if (checkCatBudget) {  
 bool checkUnused= Category.*checkUnusedBudget* (newAmount);  
 if (checkUnused) {  
 newAmount = Category.*useUnusedBudget*(newAmount);  
 }  
 else{  
 while(!checkUnused){  
 Category.*decreaseCategoryBudget*(newAmount);  
 checkUnused= Category.*checkUnusedBudget* (newAmount);  
 }  
 newAmount = Category.*useUnusedBudget*(newAmount);  
 }  
 }  
 setState(() {  
 Hive.box<Payment>('PaymentBox').putAt(  
 index,  
 Payment(  
 \_categoryController.text,  
 newAmount,  
 \_commentController.text));  
 });  
 Navigator.*of*(context).pop();  
 },  
 ),  
 ],  
 );  
 },  
 );  
 },  
 );  
 },  
 separatorBuilder: (context, index) => SizedBox(  
 height: 10,  
 )),  
 floatingActionButton: FloatingActionButton(  
 onPressed: () {  
 \_categoryController.text = '';  
 \_amountController.text = '';  
 \_commentController.text = '';  
 showDialog(  
 context: context,  
 builder: (BuildContext context) {  
 return AlertDialog(  
 title: Text('New Payment'),  
 content: Column(  
 mainAxisSize: MainAxisSize.min,  
 children: [  
 DropdownButton<String>(  
 items: categoryNames.map(buildMenuItem).toList(),  
 onChanged: (value) =>setState(() {  
 this.value=value;  
 })),  
 TextField(  
 inputFormatters: <TextInputFormatter>[  
 FilteringTextInputFormatter.allow(  
 RegExp(r'^[1-9][0-9]\*')),  
 ],  
 controller: \_amountController,  
 decoration: InputDecoration(  
 labelText: 'Amount',  
 )),  
 TextFormField(  
 controller: \_commentController,  
 validator: (value) {  
 if (value == null) {  
 return null;  
 }  
 },  
 decoration: InputDecoration(  
 labelText: 'Comment',  
 ),  
 ),  
 ],  
 ),  
 actions: [  
 TextButton(  
 child: Text('Cancel'),  
 onPressed: () {  
 Navigator.*of*(context).pop();  
 },  
 ),  
 TextButton(  
 child: Text('Save'),  
 onPressed: () {  
 double newAmount =  
 double.*parse*(\_amountController.text);  
 bool checkCatBudget = Category.*checkCategoryB*(newAmount);  
 if (checkCatBudget) {  
 bool checkUnused= Category.*checkUnusedBudget* (newAmount);  
 if (checkUnused) {  
 newAmount = Category.*useUnusedBudget*(newAmount);  
 }  
 else{  
 while(!checkUnused){  
 Category.*decreaseCategoryBudget*(newAmount);  
 checkUnused= Category.*checkUnusedBudget* (newAmount);  
 }  
 newAmount = Category.*useUnusedBudget*(newAmount);  
 }  
 }  
 setState(() {  
 Hive.box<Payment>('PaymentBox').add(Payment(  
 \_categoryController.text,  
 newAmount,  
 \_commentController.text));  
 });  
 Navigator.*of*(context).pop();  
 },  
 ),  
 ],  
 );  
 },  
 );  
 },  
 backgroundColor: Colors.*orange*,  
 child: Icon(Icons.*add*),  
 ),  
 );  
 }  
 DropdownMenuItem<String> buildMenuItem(String item) => DropdownMenuItem(  
 value: item,  
 child: Text(item),  
 );  
}

Graphical user interface, diagram

Description automatically generated**4. Pielikums**

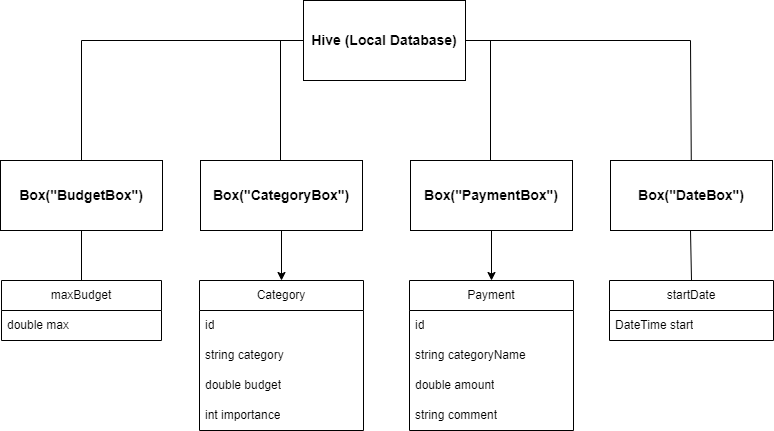
**5. Pielikums**

import 'package:flutter/cupertino.dart';  
import 'package:flutter/material.dart';  
import 'package:pie\_chart/pie\_chart.dart';  
import 'package:untitled/overviewdata.dart';  
class overviewPage extends StatefulWidget {  
 const overviewPage({Key? key}) : super(key: key);  
 @override  
 State<overviewPage> createState() => \_overviewPageState();  
}  
  
class \_overviewPageState extends State<overviewPage> {  
  
 Map<String, double> dataMap = OverviewData.*getDataMap*();  
 double rbudget = OverviewData.*getRemainingBudget*();  
 double freebudget = OverviewData.*getUnusedBudget*();  
 double maxbudget = OverviewData.*getMaxBudget*();  
 int days = OverviewData.*getDays*();  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 backgroundColor: CupertinoColors.*systemGroupedBackground*,  
 body: ListView(  
 children: [  
 Container(  
 decoration: BoxDecoration(  
 color: Colors.*orange*,  
 borderRadius: BorderRadius.all(  
 Radius.circular(30),  
 ),  
 border: Border.all(color: Colors.*orange*, width: 4)),  
 margin: EdgeInsets.fromLTRB(7, 16, 7, 0),  
 padding: EdgeInsets.all(3),  
 child: Column(  
 children: <Widget>[  
 Container(  
 decoration: BoxDecoration(  
 color: Colors.*orange*,  
 borderRadius: BorderRadius.all(Radius.circular(30))),  
 margin: EdgeInsets.fromLTRB(3, 2, 3, 0),  
 padding: EdgeInsets.all(8),  
 child: Text("Day(s) until the end:",  
 style: TextStyle(  
 fontSize: 43,  
 color: Colors.*white*,  
 fontFamily: 'mb',  
 )),  
 ),  
 Container(  
 decoration: BoxDecoration(  
 color: Colors.*white*,  
 shape: BoxShape.circle,  
 ),  
 child: Text(days.toString(),  
 style: TextStyle(  
 fontSize: 50,  
 color: Colors.*orange*,  
 fontFamily: 'mb',  
 )),  
 margin: EdgeInsets.fromLTRB(3, 0, 3, 0),  
 padding: EdgeInsets.all(20),  
 ),  
 ],  
 ),  
 ),  
 Container(  
 padding: EdgeInsets.all(8),  
 margin: EdgeInsets.fromLTRB(7, 10, 7, 0),  
 decoration: BoxDecoration(  
 color: Colors.*orange*,  
 borderRadius: BorderRadius.all(Radius.circular(30))),  
 child: Column(  
 children: [  
 Container(  
 padding: EdgeInsets.all(8),  
 child: Row(  
 mainAxisAlignment: MainAxisAlignment.center,  
 children: [  
 Container(  
 padding: EdgeInsets.only(  
 left: 0, top: 0, right: 10, bottom: 0),  
 child: Text("Remaining budget:",  
 style: TextStyle(  
 color: Colors.*white*,  
 fontSize: 32,  
 fontFamily: 'mb')),  
 ),  
 Container(  
 decoration: BoxDecoration(  
 color: Colors.*white*,  
 borderRadius: BorderRadius.all(  
 Radius.circular(20),  
 ),  
 border:  
 Border.all(color: Colors.*white*, width: 4)),  
 padding: EdgeInsets.only(  
 left: 0, top: 0, right: 0, bottom: 0),  
 child: Text(rbudget.toString() + ' \$',  
 style: TextStyle(  
 color: Colors.*orange*,  
 fontSize: 32,  
 fontFamily: 'mb')),  
 ),  
 ],  
 )),  
 Container(  
 margin: EdgeInsets.fromLTRB(0, 0, 0, 0),  
 padding: EdgeInsets.fromLTRB(0, 0, 0, 10),  
 child: Row(  
 mainAxisAlignment: MainAxisAlignment.center,  
 children: [  
 Container(  
 padding: EdgeInsets.only(  
 left: 0, top: 0, right: 0, bottom: 0),  
 child: Text("Max: ",  
 style: TextStyle(  
 color: Colors.*white*,  
 fontSize: 28,  
 fontFamily: 'mb')),  
 ),  
 Container(  
 decoration: BoxDecoration(  
 color: Colors.*white*,  
 borderRadius: BorderRadius.all(  
 Radius.circular(20),  
 ),  
 border:  
 Border.all(color: Colors.*white*, width: 4)),  
 padding: EdgeInsets.only(  
 left: 0, top: 0, right: 0, bottom: 0),  
 child: Text(maxbudget.toString() + ' \$',  
 style: TextStyle(  
 color: Colors.*orange*,  
 fontSize: 28,  
 fontFamily: 'mb')),  
 ),  
 Container(  
 padding: EdgeInsets.only(  
 left: 10, top: 0, right: 0, bottom: 0),  
 child: Text("Unused: ",  
 style: TextStyle(  
 color: Colors.*white*,  
 fontSize: 28,  
 fontFamily: 'mb')),  
 ),  
 Container(  
 decoration: BoxDecoration(  
 color: Colors.*white*,  
 borderRadius: BorderRadius.all(  
 Radius.circular(20),  
 ),  
 border:  
 Border.all(color: Colors.*white*, width: 4)),  
 padding: EdgeInsets.only(  
 left: 0, top: 0, right: 0, bottom: 0),  
 child: Text(freebudget.toString() + ' \$',  
 style: TextStyle(  
 color: Colors.*orange*,  
 fontSize: 28,  
 fontFamily: 'mb')),  
 ),  
 ],  
 ),  
 ),  
 ],  
 )),  
 Container(  
 margin: EdgeInsets.fromLTRB(0, 20, 0, 0),  
 child: PieChart(  
 dataMap: dataMap,  
 chartValuesOptions: ChartValuesOptions(  
 chartValueStyle: TextStyle(  
 fontSize: 20,  
 color: Colors.*black*,  
 fontFamily: 'mb',  
 )  
 ),  
 legendOptions: LegendOptions(  
 legendTextStyle: TextStyle(  
 fontSize: 17,  
 )  
 ),  
 ),  
 )  
 ],  
 ),  
 );  
 }  
}

**6. Pielikums**

Chart, pie chart

Description automatically generated

**7. Pielikums**