**Budžeta aplikācijas dokumentācija**

**1.Problēmas izpēte**

**1.1. Procesa izvēle un pamatojums:**

* **Lietotāju intervija:** sarunas ar diviem galvenajiem lietotājiem, lai saprastu viņu vajadzības, problēmas ar esošajām aplikācijām un vēlamās funkcijas;
* **Esošās aplikācijas analīze:** izpētīt esošās budžeta aplikācijas un identificētu to trūkumus;
* **Datu procesu novērošana:** novērot, kā lietotāji plāno budžetu, kā viņi ievada datus un kādi ir viņu galvenie izaicinājumi.

Izvēlētās problēmas izpētes izriet no aplikācijas lietotājiem, tā kā paredzētais lietotāju skaits ir tikai divi cilvēki kur viens no tiem ir es, nav iespejāms veikt lielākas aptaujas kurās tiktu izmantotas aptauju anketas. Tātad lai atpoguļotu savas un otra cilvēka vēlmes uzskatu, ka izvēloties vienkāršu interviju, pašlaik izmantotās aplikācijas analīzi un procesu novērošanu spēšu pilvērtīgi to izdarīt.

**1.2. Izpētes procesa apraksts:**

* **Datu vākšana:** izmantojot izvēlētos procesus ievākt visu nepieciešamo informāciju no galvenajiem aplikācijas lietotājiem;
* **Problēmu identificēšana:** no visiem ievāktajiem datiem atpazīt galvenās sūdzības;
* **Vēlamās aplikācijas funkcijas:** noteikt iztrūkstošās funkcijas jau izmantotajā aplikācija kuras nepieciešams ieviest;
* **Automatizācijas iespēju noteikšana:** noteikt kurus procesus ir iespējams automatizēt, spriežot pēc novērotajiem procesiem.

**1.3. Izpētes datu apkopojums:**

* **Lietotāja sūdzības:**
  1. google konta nepieciešamība - kas rada nesaistītu datu automātisko sinhronizāciju starp pievienotajām ierīcēm, pat pēc šīs sinhronizācijas izlēgšanas, piemēram telefona numuri kur tie tiek pārkopēti no vienas ierīces otrā un sakritīgo nosaukumi tiek mainīti radot apgrūtinājumu atpazīt pareizos kontaktus;
  2. datu zudumi un nepareiza saglabāšana - bieža reālā budžeta nesakritība ar to kas ir fiksēts aplikācijā;
  3. reklāmas un maksas funkcijas;
  4. lietotāja datu saglabāšana un nodošana trešajām pusēm.
* **Lietotāja vajadzības:**

1. konta izveide bez trešās puses iesaistes;
2. bez reklāmām un bezmaksas;
3. dati tiek uzglabāti tikai personīgajās ierīcēs;
4. dati tiek uzglabāti un apstrādāti pareizi;
5. paroles tiek uzglabātas drošā veidā;
6. ienākumu un Izdevumu fiksēšana (datums, summa, kategorija);
7. var redzēt visus Ienākumus un Izdevumus;
8. jaunu kategoriju izveide;
9. iespēja izdzēst ierakstus;
10. iespēja šķirot datus pēc Ienākumiem un Izdevumiem;
11. redzama kopējā bilance;
12. visi dati tiek šifrēti.

* **Automatizācijas iespējas:**
  1. sinhronizācija starp ierīcēm;
  2. datu pārnešana no citām aplikācijām;
  3. automātiska transakciju nolasīšana no bankas kontiem un pievienošana aplikācijā;
  4. fizisko čeku atpazīšana no bildēm, spēja nolasīt šos datus un automātiska ievade aplikācijā izmantojot mašīnmācīšanos.

**2. Aplikācijas prasības un funkcionalitāte**

**2.1. Mērķauditorija un tās raksturojums**

Galvenā mērķauditorija šai aplikācija ir divi cilvēki kas vēlas skaidri un efektīvi uzskaitīt savas finanses.

Šie lietotāji ir:

* **Neprofesionālis un profesionālis finansēs:** kuriem ir svarīgi sekot līdzi savām finansēm;
* **Tehnoloģiju lietotāji:** pieraduši lietot aplikāciju finanšu sekošanai, bet neapmierināti ar esošajiem piedāvājumiem;
* **Drošības apzināti:** viņiem ir svarīga sava privātā informācija, tāpēc arī vēlas lai dati būtu droši un privāti, bez trešo pušu piesaistes.

**2.2. Programmatūras un tās funkcijas apraskts**

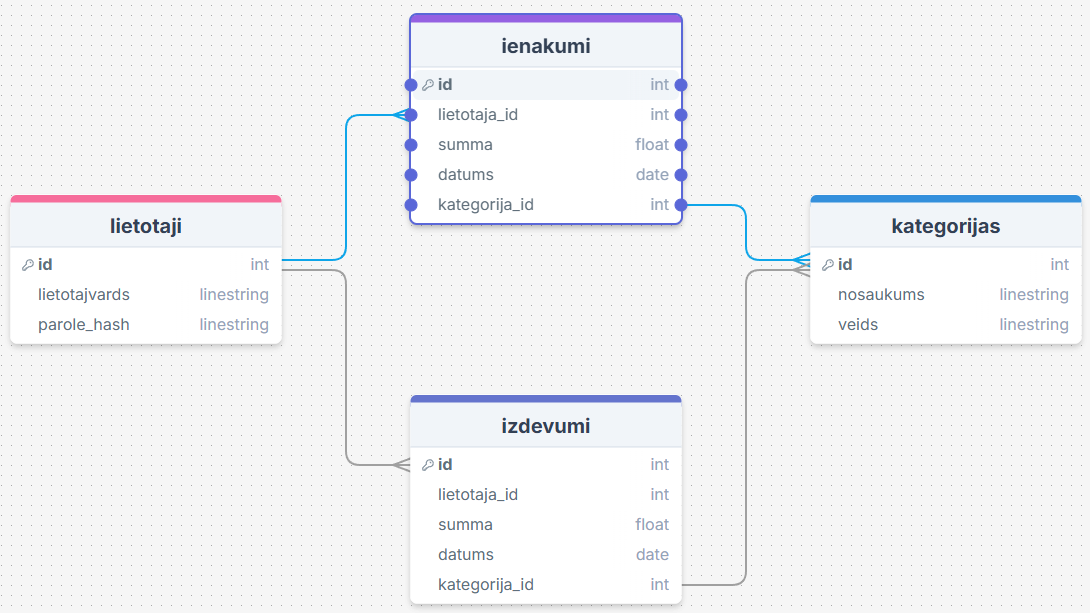
Funkcionālās (jābūt iekļautām):

1. **Lietotāja reģistrācija un autentifikācija:**
2. reģistrācija bez trešo pušu iesaistes;
3. droša autentifikācija ar šifrētu paroli;
4. var izveidot vairākus lietotāju kontus.
5. **Datu sinhronizācija un drošība:**
6. dati saglabāti lokāli ierīcēs;
7. nav reklāmu un maksas funkciju.
8. **Budžeta pārvaldība:**
9. var pievienot jaunus Ienākumus un Izdevumus,kuriem klāt jābūt ir datumam, summai un kategorijai;
10. spēja izdzēst ierakstus.
11. **Pārskati un analīze:**
12. ienākumu un izdevumu pārskatu var uzrādīt atsevišķi vai kopēji;
13. ieraksti tiek sagrupēti pēc datumiem dilstošā secībā.

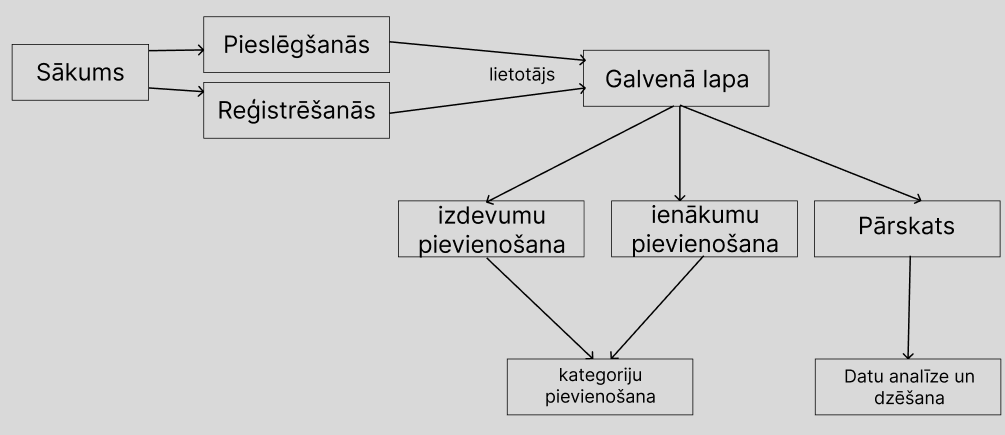
**2.3. Nefunkcionālās (nepieciešams ieviest vēlāk):**

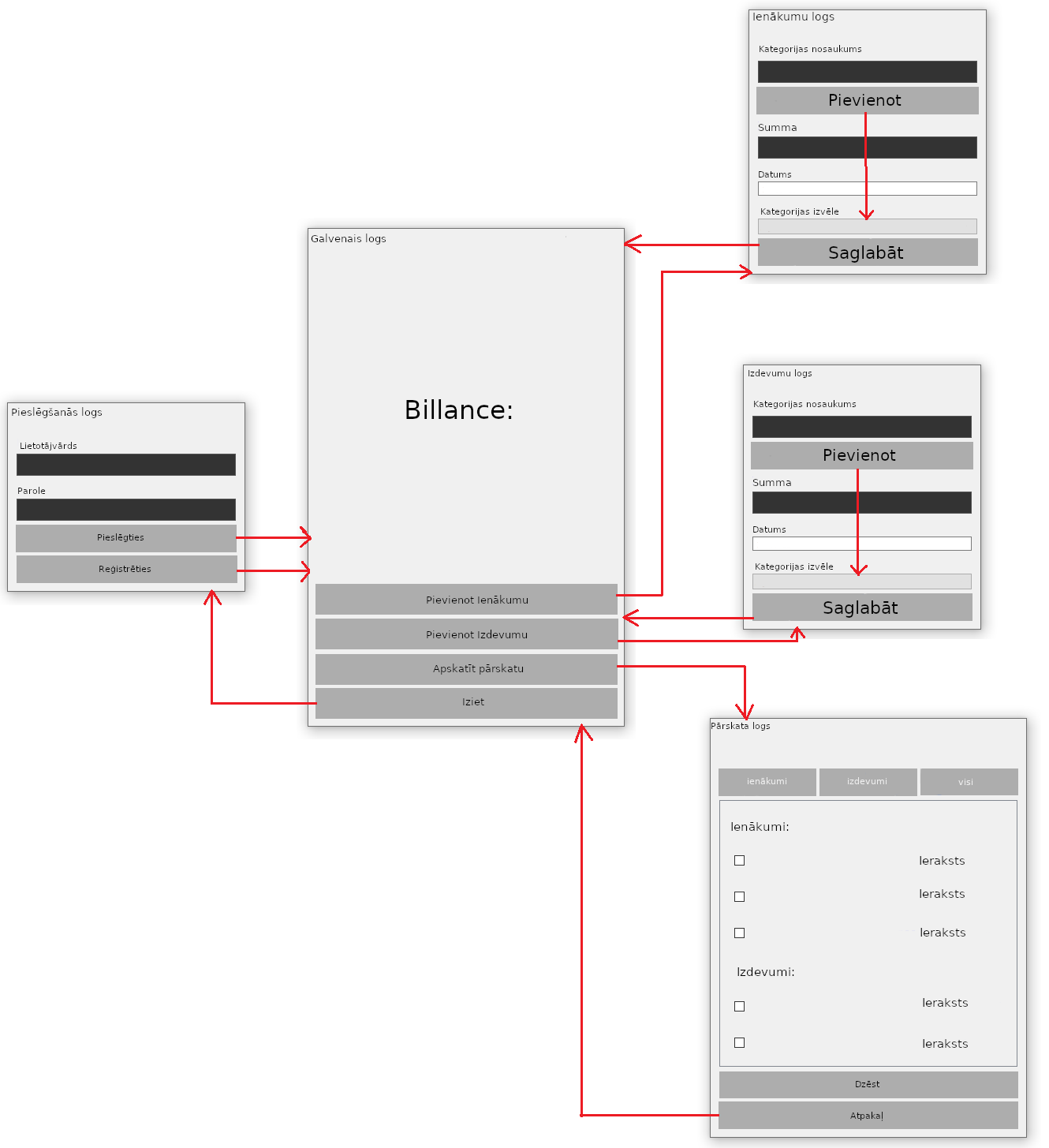
* Šifrētu datu sinhronizācija starp divām ierīcēm bez trešo pušu pakalpojumiem;
* Visi dati tiek šifrēti;
* Bankas konta piesaiste automātiskai transakciju ievadīšanai aplikācijā;
* Iespēja pievienot vairākus budžeta kontus vienam lietotājam (galvenais, iekrājumu utt.);
* Pielāgojamas kategorijas ar ikonam;
* Var pievienot komentārus ierakstiem;
* Var rediģēt un nokopēt ierakstus;
* Spēja atzīmēt svarīgos ierakstus un ikmēneša maksājumus;
* Ikmēneša maksājumi notiek automātiski;
* Iespēja izvēlēties valūtu budžeta kontam;
* Spēja pievienot datus un tos automātiski sagrupēt, pielietojot mašīnmācīšanos, attiecīgajās kategorijās izmantojot fotoattēlu ar čeku;
* Iespēja šķirot ierakstus pārskatā pēc dienām, nedēļām, mēnešiem, gadiem un izvēlētā perioda;
* Pie katra ieraksta uzrādīt cik, procentuāli, no budžeta tas aizņem, kā arī kopējo summu, kategoriju, datumu un pogas lai rediģētu, dzēstu vai atzīmētu ierakstu;
* Atgādinājumi par ikmēneša maksājumiem;
* Kalendārs kur var atzīmēt svarīgas dienas vai izdevumus un pievienot komentārus. (piem. Svētku dienas).

**2.4. Datu bāzes projektējums:**

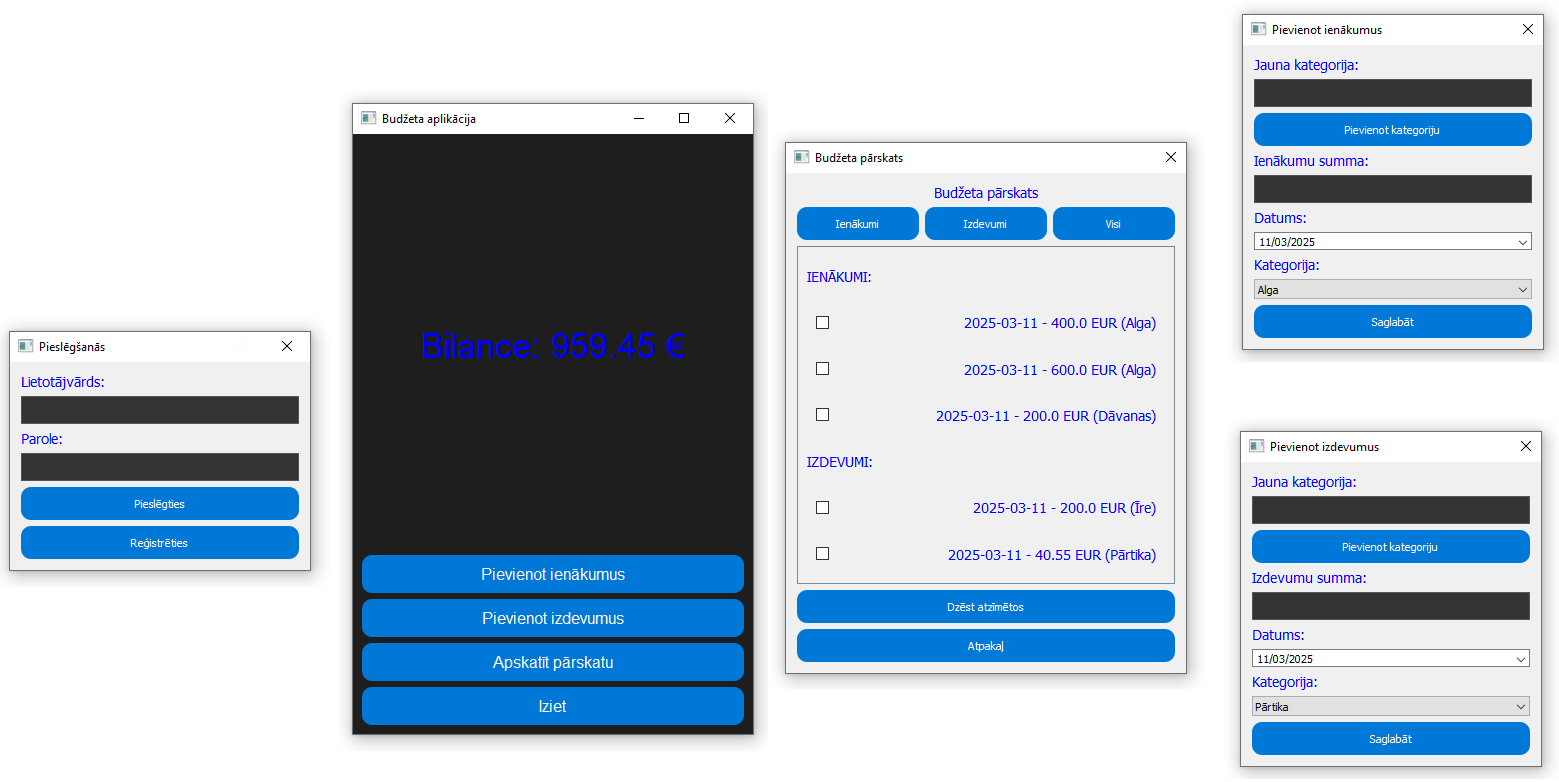


**2.5. Saskarņu īpašības:**



**2.6. Saskarnes karkasa diagramma un skice:**

**2.7. Dizains:**



**2.8. Izvēlētā valoda un vide:**

Aplikācija tiks izstrādāta, izmantojot Python kā galveno programmēšanas valodu, bet izmantošu arī papildus valodas un bibliotēkas pilnu funkciju ieviešanai. Izvēlējos Python jo tā nodrošina plašu bibliotēku klāstu un labu atbalstu datu apstrādei kā arī mašīnmācīšanās algoritmus. Izstrādes vide būs Visual Studio Code.

**Galvenā valoda:** Python.

**Papildu valodas/tehnoloģijas:**

1. SQL (datu bāzes pārvaldībai).
2. Bash (automatizācijai).
3. JSON (datu apmaiņai).
4. PyQt5 (grafiskajai saskarsnei)
5. SQLAlchemy (rīks lai atvieglotu darbu ar datubāzi)
6. Bcrypt (paroļu šifrēšanai)
7. Github (versiju izmitināšanai)

**3. Izstrādes plāns**

**3.1. Izstrādes modelis:**

Aplikācijas izstrādei es izvēlējos ātrās prototipēšanas modeli jeb Agile, jo tas ļaus pakāpeniski pilnveidot aplikāciju, balstoties uz testēšanas rezultātiem. Vieglāk un ātrāk pārbaudīt, un veikt labojumus vai papildinājums pēc funkciju testēšanas.

**3.2. Izstrādes posmi:**

**1. posms:** **Pamatfunkcionalitātes izstrāde**

1. Lietotāja reģistrācija un autentifikācija:
2. Izveidot reģistrācijas un autentifikācijas sistēmu.
3. Implementēt drošu paroļu šifrēšanu, izmantojot Bcrypt.
4. Datu bāzes izveide: izveidot SQLite datu bāzi ar tabulām lietotājiem, ienākumiem, izdevumiem un kategorijām.
5. Budžeta pārvaldība: ieviest iespēju pievienot, dzēst un apskatīt ienākumus un izdevumus.
6. Dizaina izstrāde:izveidot kopīgu dizainu aplikācijai.

**2. posms: Pārskatu un analīzes funkcionalitāte**

1. Pārskatu izveide:
2. Ieviest iespēju apskatīt ienākumus un izdevumus atsevišķi vai kopēji.
3. Ierakstu grupēšana pēc datuma dilstošā secībā.

**3. posms: Nefunkcionālo prasību ieviešana**

1. Datu drošība: ieviest pilnu datu šifrēšanu.
2. Papildu funkcionalitātes:
3. Ierakstu rediģēšanas un kopēšanas funkcionalitāte;
4. Komentāru pievienošana pie ierakstiem;
5. Procentuālā vizualizācija;
6. Pielāgojamas kategorijas ar ikonām;
7. Valūtas izvēle;
8. Ierakstu šķirošana;
9. Kalendāra ieviešana;
10. Atgādinājumi par maksājumiem;
11. Ierakstu atzīmēšanas funkcija;
12. Vairāku budžeti lietotāja pārvaldība;
13. Bankas konta piesaiste un automātiska transakciju ievade;
14. Automātiskie maksājumi;
15. Mašīnmācīšanās integrācija datu grupēšanai;
16. Android aplikācijas izveide.

**3.3. Testēšana, dokumentēšana un izlaišana**

1. Testēšana:
2. Veikt funkcionālos un drošības testus katrā izstrādes posmā;
3. Lietotāju testēšana ar diviem galvenajiem lietotājiem.
4. Dokumentēšana: programmas dokumentācijas izveide.
5. Izlaišana:
6. Sagatavot un publicēt pirmo stabilo versiju;
7. Nodrošināt lietotāja rokasgrāmatu;
8. Veikt pēc-izlaišanas uzraudzību un uzlabojumus.

**3.4. Uzturēšana un uzlabojumi**

* Ievākt lietotāju atsauksmes un identificēt iespējamās uzlabojumu jomas;
* Veikt regulārus drošības atjauninājumus;
* Attīstīt jaunas funkcijas, balstoties uz lietotāju vajadzībām.

**4. Atkļūdošanas un akceptēšanas pārskats**

**1. Atkļūdošanas process**

**1.1. Metodoloģija**

Atkļūdošanas process tika veikts manuāli, analizējot programmas darbību dažādos scenārijos, kā arī ņemot vērā lietotāju atsauksmes. Pēc katra identificētā trūkuma tika veikta tā labošana un atkārtota testēšana, lai pārliecinātos par izmaiņu efektivitāti un sistēmas stabilitāti.

**1.2. Galvenās problēmas un to novēršana**

**1.2.1. Nepareiza bilances aprēķināšana un noapaļošana**

Problēma: Bilance netika aprēķināta pareizi un netika noapaļota līdz divām decimāldaļām.

Risinājums: Pievienota .2f formatēšana, lai nodrošinātu precīzu noapaļošanu līdz diviem cipariem aiz komata.

**1.2.2. Kategoriju manuāla ievade**

Problēma: Lietotājiem bija jāievada katra ieraksta kategorija manuāli, nebija iespējas izvēlēties no iepriekš saglabātām kategorijām.

Risinājums: Izveidota jauna datubāzes tabula kategoriju glabāšanai un attiecīgas funkcijas, kas saglabā un ielādē kategorijas automātiski no datubāzes.

**1.2.3. Nepareiza datuma izvēle**

Problēma: Lietotāji varēja izvēlēties nākotnes datumus, kas nav vēlama funkcionalitāte budžeta pārvaldības sistēmā.

Risinājums: Pievienota datuma ierobežošana ar self.datums\_ievade.setMaximumDate(QDate.currentDate()), kas nodrošina, ka var izvēlēties tikai pašreizējo vai iepriekšējos datumus.

**1.2.4. Bilance netika atjaunota pēc ierakstu dzēšanas**

Problēma: Pēc ierakstu dzēšanas bilance netika automātiski atjaunināta, kā rezultātā tika attēlota nepareiza informācija.

Risinājums: Pievienots self.bilance\_label.setText(self.lasi\_bilanci()), kas nodrošina bilances automātisku atjaunināšanu pēc ierakstu izmaiņām.

**1.2.5. Nepareiza lietotāja sesijas pārvaldība**

Problēma: Pēc iziešanas no konta un atkārtotas pieslēgšanās ar citu kontu sistēma atvēra iepriekšējā lietotāja informāciju.

Risinājums: Izveidota funkcija iziet\_no\_konta(self), kas aizver pašreizējo logu un pēc veiksmīgas pieslēgšanās restartē galveno logu ar jaunā lietotāja datiem.

**2. Akceptēšanas kritēriji**

Lai budžeta aplikācija tiktu akceptēta, tai bija jāatbilst šādiem kritērijiem:

* Bilance tiek pareizi aprēķināta un attēlota ar divām zīmēm aiz komata;
* Kategorijas tiek automātiski ielādētas no datubāzes un piedāvātas izvēlei;
* Datumu ievades lauks neļauj izvēlēties nākotnes datumus;
* Pēc ierakstu dzēšanas bilance tiek nekavējoties atjaunināta;
* Sistēma pareizi pārvalda lietotāju sesijas un nerāda iepriekšējā lietotāja informāciju pēc atkārtotas pieslēgšanās;
* Lietotājs var veiksmīgi reģistrēties un autentificēties;
* Paroles tiek šifrētas ar Bcrypt un nav pieejamas atklātā veidā datubāzē;
* Funkcionalitāte darbojas bez kļūdām;
* Struktūra korekta, un visi ieraksti tiek veiksmīgi saglabāti un ielādēti;
* Iespēja pievienot, dzēst un apskatīt ienākumus un izdevumus;
* Ieraksti tiek veiksmīgi saglabāti, dzēsti un parādīti lietotāja interfeisā;
* Programma saglabā vienotu dizaina stilu, interfeiss ir lietotājam draudzīgs;
* Lietotājs var apskatīt ienākumus un izdevumus atsevišķi vai kopēji;
* Pārskati attēlo datus korekti un vizuāli skaidri;
* Visi ieraksti automātiski tiek sakārtoti pēc datuma (no jaunākajiem uz vecākajiem).

**3. Testēšanas rezultāti**

Testēšana tika veikta, izpildot dažādus lietošanas scenārijus.

Galvenie rezultāti:

|  |  |
| --- | --- |
| **Testa scenārijs** | **Rezultāts** |
| Bilance tiek pareizi aprēķināta un attēlota ar divām zīmēm aiz komata. | Veiksmīgi |
| Kategorijas tiek automātiski ielādētas no datubāzes un piedāvātas izvēlei. | Veiksmīgi |
| Datumu ievades lauks neļauj izvēlēties nākotnes datumus. | Veiksmīgi |
| Pēc ierakstu dzēšanas bilance tiek nekavējoties atjaunināta. | Veiksmīgi |
| Sistēma pareizi pārvalda lietotāju sesijas un nerāda iepriekšējā lietotāja informāciju pēc atkārtotas pieslēgšanās. | Veiksmīgi |
| Lietotājs var veiksmīgi reģistrēties un autentificēties. | Veiksmīgi |
| Paroles tiek šifrētas ar Bcrypt un nav pieejamas atklātā veidā datubāzē. | Veiksmīgi |
| Funkcionalitāte darbojas bez kļūdām. | Veiksmīgi |
| Struktūra korekta, un visi ieraksti tiek veiksmīgi saglabāti un ielādēti. | Veiksmīgi |
| Iespēja pievienot, dzēst un apskatīt ienākumus un izdevumus. | Veiksmīgi |
| Ieraksti tiek veiksmīgi saglabāti, dzēsti un parādīti lietotāja interfeisā. | Veiksmīgi |
| Programma saglabā vienotu dizaina stilu, interfeiss ir lietotājam draudzīgs. | Veiksmīgi |
| Lietotājs var apskatīt ienākumus un izdevumus atsevišķi vai kopēji. | Veiksmīgi |
| Pārskati attēlo datus korekti un vizuāli skaidri. | Veiksmīgi |
| Visi ieraksti automātiski tiek sakārtoti pēc datuma (no jaunākajiem uz vecākajiem). | Veiksmīgi |

**4. Lietotāju atsauksmes un pēdējie labojumi**

Testēšanā piedalījās vairāki lietotāji, kuri sniedza šādas atsauksmes:

* Pozitīvi novērtēta iespēja izvēlēties kategorijas no saraksta;
* Bilances atjaunošana pēc ierakstu dzēšanas uzlabojusi lietojamības pieredzi;
* Lietotāju sesiju pareiza pārvaldība nodrošinājusi labāku drošību un ērtību.

Pēc lietotāju atsauksmēm tika veikti nelieli uzlabojumi interfeisā, lai padarītu sistēmu intuitīvāku un vieglāk saprotamu.

**5. Akceptēšanas lēmums**

Pēc atkļūdošanas un akceptēšanas testi apliecina, ka programma atbilst sākotnējās versijas prasībām. Visi būtiskie defekti ir novērsti, un pamatfunkcionalitāte darbojas stabili. Programma ir gatava turpmākajai attīstībai un testēšanai reālās lietošanas vidē.

**5. Lietotāja ceļvedis**

**Lietotāju konta izveide:**

Visas sistēmas funkcijas ir pieejamas katram lietotājam kurš ir reģistrējies. Reģistrācijai ir nepieciešams unikāls lietotājvārds un parole, juri jāievada tiem norādītajos ievades laukos. Pēc lietotājvārda un paroles ievades jānospiež poga “Reģistrēties” kas saglabās ievadītos datus datubāzē un automātiski novirzīs uz galveno aplikācijas skatu.

**Lietotāju autentificēšanās:**

Ja lietotājs jau ir reģistrējies, viņam ir nepieciešams ievadīt lietotājvārdu un paroli ar kuriem ir reģistrēts aplikācija, tiem paredzētajos laukos un nospiest pogu “Pieslēgties” kas pārbaudīs vai lietotājs ir reģistrēts un pēc apstiprināšanas novirzīs to uz galveno lapu.

**Ienākumu, izdevumu pievienošana un kategorijas izveide:**

* Lai varētu ievadīt jaunu ienākumu vai izdevumu, galvenajā lapā ir nepieciešams nospiest pogu “Pievienot ienākumu” vai “Pievienot izdevumu” kas atvērs attiecīgos logus datu ievadei.
* Pirms ienākuma vai izdevumu pievienošanas, sākumā japievieno kategoriju, lai to izdarītu nepieciešams ievadīt kategorijas nosaukumu ievades laukā zem “Jauna kategorija” un nospiest pogu “Pievienot kategoriju”
* Pēctam pievienotā kategorija būs pieejama visiem nāktones ienākumiem un izdevumiem izvēles sarakstā “Kategorija:”
* Tālāk var pievienot jaunu Ienākumu vai izdevumu. Jāievada summa, tad jāizvēlas datumu un kategoriju, kad tas ir izdarīts tad jānospiež poga “Saglabāt” kas saglabās ievadītos datus un novirzīs atpakaļ uz galveno lapu.
* Ja nevēlies pievienot jaunu ierakstu tad var nospiest simbolu x kas atrodas loga augšejā labējā stūrī.

**Pārskata apskatīšana:**

* Lai varētu apskatīt pārskatu kur ir redzami visi ienākumi un izdevumi, jāspiež uz pogas “Apskatīt pārskatu” kas atvērs jaunu logu.
* Tagad kad ir atvērta pārskata lapa ir redzami visi veiktie ieraksti kas ir sakārtoti dilstošā secībā pēc datumiem sākot ar jaunāko datumu.
* Pogas “Ienākumi” “Izdevumi” “Visi” atbild par datu atlasi. Nospiežot šīs pogas pārskatā tiek parādīti tikai ienākumi, tikai izdevumi vai visi ieraksti.
* Nospiežot un atķeksējot mazos lodziņus pie ierakstiem un tad nospiežot pogu “Dzēst atzīmētos” tiek neatgriezeniski izdzēsti visi atzīmētie ieraksti.
* Nospiežot pogu “Atpakaļ” tiks aizvērts Pārskata logs un lietotājs tiks novirzīts atpakaļ uz galveno lapu.

**Iziešana no konta:**

Lai izietu no konta ir jānospiež poga “Iziet”, tad kad tas ir izdarīts tiks atvērts pieslēgšanās logs, no kura var pieslēgties vai reģistrēt jaunu lietotāju.

**Aplikācijas aizvēršana:**

Lai aizvērtu aplikāciju pilnībā ir jāuzklikšķina uz simbolu x kas atrodas logu labajā augšējā stūrī. To ir iespējams izdarīt no jebkura loga izņemot ienākumu un izdevumu pievienošanas logiem.

*Pielikums*

Bilance.py

from dati.datubaze import Ienakumi, Izdevumi, sesija

def generet\_parskatu(lietotaja\_id):

ienakumi = sesija.query(Ienakumi).filter\_by(lietotaja\_id=lietotaja\_id).all() # Iegūst visus ienākumus konkrētajam lietotājam no datubāzes

izdevumi = sesija.query(Izdevumi).filter\_by(lietotaja\_id=lietotaja\_id).all() # Iegūst visus izdevumus konkrētajam lietotājam no datubāzes

ienakumu\_summa = sum(i.summa for i in ienakumi) # Aprēķina kopējo ienākumu summu

izdevumu\_summa = sum(i.summa for i in izdevumi) # Aprēķina kopējo izdevumu summu

bilance = ienakumu\_summa - izdevumu\_summa # Aprēķina bilanci (ienākumi - izdevumi)

return f"Ienākumi: {ienakumu\_summa:.2f}, Izdevumi: {izdevumu\_summa:.2f}, Bilance: {bilance:.2f}" # Atgriež vērtības ar diviem cipariem aiz komata

Datubaze.py

from sqlalchemy import create\_engine, Column, Integer, String, Float, Date, ForeignKey

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

import bcrypt

import os

# Norāda kur tiks sagalbāta datubāze un ar kādu nosaukumu

datu\_mape = "dati"

db\_cels = os.path.join(datu\_mape, "budzets.db")

# Datubāzes savienojums

engine = create\_engine(f"sqlite:///{db\_cels}", echo=False)

# SQLAlchemy bāzes klase modeļu definēšanai

baze = declarative\_base()

# Lietotāju tabula

class Lietotajs(baze):

\_\_tablename\_\_ = "lietotaji" # Tabulas nosaukums

id = Column(Integer, primary\_key=True) # Primārā atslēga

lietotajvards = Column(String, unique=True, nullable=False) # Lietotājvārds

parole\_hash = Column(String, nullable=False) # Paroles

# Metode paroles šifrēšanai un saglabāšanai

def uzstadit\_paroli(self, parole):

self.parole\_hash = bcrypt.hashpw(parole.encode("utf-8"), bcrypt.gensalt()).decode("utf-8")

# Metode paroles pārbaudei

def parbaud\_paroli(self, parole):

return bcrypt.checkpw(parole.encode("utf-8"), self.parole\_hash.encode("utf-8"))

# Ienākumu tabula

class Ienakumi(baze):

\_\_tablename\_\_ = "ienakumi" # Tabulas nosaukums

id = Column(Integer, primary\_key=True) # Primārā atslēga

lietotaja\_id = Column(Integer, ForeignKey("lietotaji.id"), nullable=False) # Ārējā atslēga

summa = Column(Float, nullable=False) # Ienākuma summa

datums = Column(Date, nullable=False) # Ienākuma datums

kategorija\_id = Column(Integer, ForeignKey("kategorijas.id"), nullable=False) # Kategorijas atslēga

# Izdevumu tabula

class Izdevumi(baze):

\_\_tablename\_\_ = "izdevumi" # Tabulas nosaukums

id = Column(Integer, primary\_key=True) # Primārā atslēga

lietotaja\_id = Column(Integer, ForeignKey("lietotaji.id"), nullable=False) # Ārējā atslēga

summa = Column(Float, nullable=False) # Izdevuma summa

datums = Column(Date, nullable=False) # Izdevuma datums

kategorija\_id = Column(Integer, ForeignKey("kategorijas.id"), nullable=False) # Kategorijas atslēga

class Kategorija(baze): # Kategoriju tabula

\_\_tablename\_\_ = "kategorijas" # Tabulas nosaukums

id = Column(Integer, primary\_key=True) # Primārā atslēga

nosaukums = Column(String, unique=True, nullable=False) # Kategorijas nosaukums

veids = Column(String, nullable=False) # Kategorijas veids (ienākumi/izdevumi)

# Izvedio tabulas, ja tādas nepastāv

baze.metadata.create\_all(engine)

# Sesijas inicializācija

Sesija = sessionmaker(bind=engine)

sesija = Sesija()

Stils.py

def get\_stils():

return """

QMainWindow {

background-color: #1E1E1E;

}

QLabel {

color: blue;

font-size: 14px;

}

QPushButton {

background-color: #0078D7;

color: white;

border-radius: 10px;

padding: 10px;

}

QPushButton:hover {

background-color: #005A9E;

}

QLineEdit {

background-color: #333;

color: white;

border: 1px solid #555;

padding: 5px;

}

"""

Galvenais\_logs.py

import sys

from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton, QLabel, QVBoxLayout, QWidget, QDialog

from PyQt5.QtGui import QFont

from PyQt5.QtCore import Qt

from dizains.stils import get\_stils

from logi.izdevumu\_logs import IzdevumuLogs

from logi.ienakumu\_logs import IenakumuLogs

from dati.bilance import generet\_parskatu

from logi.parskata\_logs import ParskataLogs

from logi.pieslegsanas\_logs import PieslegsanasLogs

# Galvenā loga klase

class BudzetaAplikacija(QMainWindow):

def \_\_init\_\_(self, lietotaja\_id): # Konstruktors

super().\_\_init\_\_() # Superklases konstruktors

self.lietotaja\_id = lietotaja\_id # Saglabā lietotāja id

self.setWindowTitle("Budžeta aplikācija") # Loga nosaukums

self.setGeometry(100, 100, 400, 600) # Loga izmēri

self.setStyleSheet(get\_stils()) # Iestata dizaina stilus

self.init\_ui() # Izsauc funkciju init\_ui

def init\_ui(self):

layout = QVBoxLayout() # Izveido vertikālu izkārtojumu

# Izveido teksta logu ar bilanci

self.bilance\_label = QLabel(self.lasi\_bilanci(), self) # Iegūst bilances vērtību un izveido teksta logu

self.bilance\_label.setAlignment(Qt.AlignCenter) # Teksta loga centralizēšana

self.bilance\_label.setFont(QFont("Arial", 12)) # Teksta loga fonta izmērs

self.bilance\_label.setStyleSheet("font-size: 35px;") # Teksta loga fonta izmērs

layout.addWidget(self.bilance\_label) # Pievieno teksta logu izkārtojumam

# Poga ienākumu pievienošanai

self.ienakumi\_poga = QPushButton("Pievienot ienākumus", self) # Izveido pogu

self.ienakumi\_poga.clicked.connect(self.atvert\_ienakumu\_logu) # Pievieno pogai funkciju

# Poga izdevumu pievienošanai

self.izdevumi\_poga = QPushButton("Pievienot izdevumus", self) # Izveido pogu

self.izdevumi\_poga.clicked.connect(self.atvert\_izdevumu\_logu) # Pievieno pogai funkciju

# Poga budžeta pārskata apskatei

self.parskats\_poga = QPushButton("Apskatīt pārskatu", self) # Izveido pogu

self.parskats\_poga.clicked.connect(self.paradi\_parskatu) # Pievieno pogai funkciju

# Poga iziešanai no konta

self.iziet\_poga = QPushButton("Iziet", self) # Izveido pogu

self.iziet\_poga.clicked.connect(self.iziet\_no\_konta) # Pievieno pogai funkciju

# Pievieno visas pogas izkārtojumam

for poga in [self.ienakumi\_poga, self.izdevumi\_poga, self.parskats\_poga, self.iziet\_poga]: # Pārbauda visas pogas

poga.setFont(QFont("Arial", 12)) # Pogas fonta izmērs

layout.addWidget(poga) # Pievieno pogu izkārtojumam

# Izveido centrālo konteineru un piešķir tam izkārtojumu

konteineris = QWidget() # Izveido konteineru

konteineris.setLayout(layout) # Piešķir konteineram izkārtojumu

self.setCentralWidget(konteineris) # Izveido centrālo konteineru

# Atver ienākumu logu

def atvert\_ienakumu\_logu(self):

self.ienakumu\_logs = IenakumuLogs(self.lietotaja\_id) # Izveido ienākumu logu

self.ienakumu\_logs.exec\_() # Atver logu

self.bilance\_label.setText(self.lasi\_bilanci()) # Atjauno bilances tekstu

# Atver izdevumu logu

def atvert\_izdevumu\_logu(self): # Izveido funkciju atvert\_izdevumu\_logu

self.izdevumu\_logs = IzdevumuLogs(self.lietotaja\_id) # Izveido izdevumu logu

self.izdevumu\_logs.exec\_() # Atver logu

self.bilance\_label.setText(self.lasi\_bilanci()) # Atjauno bilances tekstu

# Atgriež bilances vērtību

def lasi\_bilanci(self): # Izveido funkciju lasi\_bilanci

dati\_bilance = generet\_parskatu(self.lietotaja\_id) # Iegūst bilances datus pēc pieslēgušos lietotāja

return f"{dati\_bilance.split(", ")[-1]} €" # Atgriež tikai bilances vērtību

# Atver budžeta pārskata logu

def paradi\_parskatu(self): # Izveido funkciju paradi\_parskatu

self.hide() # Paslēpj galveno logu

self.parskata\_logs = ParskataLogs(self.lietotaja\_id, self) # Izveido pārskata logu

self.parskata\_logs.exec\_() # Atver logu

self.bilance\_label.setText(self.lasi\_bilanci()) # Atjauno bilances tekstu

def iziet\_no\_konta(self): # Izveido funkciju iziet\_no\_konta

self.close() # Aizver pašreizējo galveno logu

pieslegsanas\_logs = PieslegsanasLogs() # Izveido pieslēgšanās logu

if pieslegsanas\_logs.exec\_() == QDialog.Accepted: # Ja lietotājs veiksmīgi pieslēdzas

self.\_\_init\_\_(pieslegsanas\_logs.lietotaja\_id) # Restartē galveno logu ar jaunu lietotāju

self.show() # Parāda jauno galveno logu

if \_\_name\_\_ == "\_\_main\_\_": # Pārbauda vai fails tiek izpildīts tieši

app = QApplication(sys.argv) # Izveido aplikācijas instanci

pieslegsanas\_logs = PieslegsanasLogs() # Izveido pieslēgšanās logu

if pieslegsanas\_logs.exec\_() == QDialog.Accepted: # Ja lietotājs veiksmīgi pieslēdzas

lietotaja\_id = pieslegsanas\_logs.lietotaja\_id # Saglabā lietotāja id

galvenais\_logs = BudzetaAplikacija(lietotaja\_id, pieslegsanas\_logs) # Izveido galveno logu

galvenais\_logs.show() # Parāda galveno logu

sys.exit(app.exec\_()) # Iziet no aplikācijas

Ienakumu\_logs.py

from PyQt5.QtWidgets import QDialog, QLabel, QLineEdit, QPushButton, QVBoxLayout, QMessageBox, QDateEdit, QComboBox

from PyQt5.QtCore import QDate, Qt

from dati.datubaze import Ienakumi, Kategorija, sesija

from dizains.stils import get\_stils

class IenakumuLogs(QDialog): # Ienākumu pievienošanas logs

def \_\_init\_\_(self, lietotaja\_id): # Inicializācija

super().\_\_init\_\_() # Superklases inicializācija

self.lietotaja\_id = lietotaja\_id # Lietotāja ID

self.setWindowTitle("Pievienot ienākumus") # Loga nosaukums

self.setWindowFlags(self.windowFlags() & ~Qt.WindowContextHelpButtonHint) # Loga pogas

self.setGeometry(150, 150, 300, 250) # Loga izmērs

self.setStyleSheet(get\_stils()) # Loga stils

self.init\_ui() # Loga izveide

def init\_ui(self):

layout = QVBoxLayout()

self.summa\_label = QLabel("Ienākumu summa:")

self.summa\_ievade = QLineEdit()

self.datums\_label = QLabel("Datums:")

self.datums\_ievade = QDateEdit()

self.datums\_ievade.setCalendarPopup(True)

self.datums\_ievade.setDate(QDate.currentDate())

self.datums\_ievade.setMaximumDate(QDate.currentDate())

self.jauna\_kategorija\_label = QLabel("Jauna kategorija:")

self.jauna\_kategorija\_ievade = QLineEdit()

self.pievienot\_kategoriju\_poga = QPushButton("Pievienot kategoriju")

self.pievienot\_kategoriju\_poga.clicked.connect(self.pievienot\_kategoriju)

layout.addWidget(self.jauna\_kategorija\_label)

layout.addWidget(self.jauna\_kategorija\_ievade)

layout.addWidget(self.pievienot\_kategoriju\_poga)

self.kategorija\_label = QLabel("Kategorija:")

self.kategorija\_ievade = QComboBox()

self.ieladet\_kategorijas()

self.pievienot\_poga = QPushButton("Saglabāt")

self.pievienot\_poga.clicked.connect(self.saglabat\_ienakumu)

layout.addWidget(self.summa\_label)

layout.addWidget(self.summa\_ievade)

layout.addWidget(self.datums\_label)

layout.addWidget(self.datums\_ievade)

layout.addWidget(self.kategorija\_label)

layout.addWidget(self.kategorija\_ievade)

layout.addWidget(self.pievienot\_poga)

self.setLayout(layout)

def pievienot\_kategoriju(self):

jaunais\_nosaukums = self.jauna\_kategorija\_ievade.text().strip() # Iegūst jaunās kategorijas nosaukumu

if not jaunais\_nosaukums:

QMessageBox.warning(self, "Kļūda", "Kategorijas nosaukums nevar būt tukšs!")

return

# Pārbauda, vai kategorija jau eksistē

ekzistejosa = sesija.query(Kategorija).filter\_by(nosaukums=jaunais\_nosaukums, veids="ienakumi").first()

if ekzistejosa:

QMessageBox.warning(self, "Kļūda", "Šāda kategorija jau pastāv!")

return

# Izveido jaunu kategoriju un saglabāt datubāzē

jauna\_kategorija = Kategorija(nosaukums=jaunais\_nosaukums, veids="ienakumi")

sesija.add(jauna\_kategorija)

sesija.commit()

# Atjaunina kategoriju izvēlni

self.ieladet\_kategorijas()

# Notīra ievades lauku

self.jauna\_kategorija\_ievade.clear()

def ieladet\_kategorijas(self):

kategorijas = sesija.query(Kategorija).filter\_by(veids="ienakumi").all() # Atrod visas kategorijas šķirojot pēc veida

for kat in kategorijas: # iterē cauri sarakstam 'kategorijas' un pievieno katru kategoriju izvēles laukam kat ir objekts no saraksta

self.kategorija\_ievade.addItem(kat.nosaukums, kat.id) # Pievieno kategoriju izvēles laukam

def saglabat\_ienakumu(self):

try:

summa = float(self.summa\_ievade.text())

datums = self.datums\_ievade.date().toPyDate() # Pārveido QDate objektu uz Python datetime objektu

kategorija\_id = self.kategorija\_ievade.currentData() # Atgriež ID, kas ir pievienots kategorijai

if summa <= 0:

raise ValueError("Summai jābūt pozitīvai!")

jauns\_ienakums = Ienakumi(lietotaja\_id=self.lietotaja\_id, summa=summa, datums=datums, kategorija\_id=kategorija\_id)

sesija.add(jauns\_ienakums)

sesija.commit()

self.close()

except ValueError as e:

QMessageBox.warning(self, "Kļūda", str(e))

Izdevumu\_logs.py

from PyQt5.QtWidgets import QDialog, QLabel, QLineEdit, QPushButton, QVBoxLayout, QMessageBox, QDateEdit, QComboBox

from PyQt5.QtCore import QDate, Qt

from dati.datubaze import Izdevumi, Kategorija, sesija

from dizains.stils import get\_stils

class IzdevumuLogs(QDialog):

def \_\_init\_\_(self, lietotaja\_id):

super().\_\_init\_\_()

self.lietotaja\_id = lietotaja\_id

self.setWindowTitle("Pievienot izdevumus")

self.setWindowFlags(self.windowFlags() & ~Qt.WindowContextHelpButtonHint)

self.setGeometry(150, 150, 300, 250)

self.setStyleSheet(get\_stils())

self.init\_ui()

def init\_ui(self):

layout = QVBoxLayout()

self.summa\_label = QLabel("Izdevumu summa:")

self.summa\_ievade = QLineEdit()

self.datums\_label = QLabel("Datums:")

self.datums\_ievade = QDateEdit()

self.datums\_ievade.setCalendarPopup(True)

self.datums\_ievade.setDate(QDate.currentDate())

self.datums\_ievade.setMaximumDate(QDate.currentDate())

self.jauna\_kategorija\_label = QLabel("Jauna kategorija:")

self.jauna\_kategorija\_ievade = QLineEdit()

self.pievienot\_kategoriju\_poga = QPushButton("Pievienot kategoriju")

self.pievienot\_kategoriju\_poga.clicked.connect(self.pievienot\_kategoriju)

layout.addWidget(self.jauna\_kategorija\_label)

layout.addWidget(self.jauna\_kategorija\_ievade)

layout.addWidget(self.pievienot\_kategoriju\_poga)

self.kategorija\_label = QLabel("Kategorija:")

self.kategorija\_ievade = QComboBox()

self.ieladet\_kategorijas()

self.pievienot\_poga = QPushButton("Saglabāt")

self.pievienot\_poga.clicked.connect(self.saglabat\_izdevumu)

layout.addWidget(self.summa\_label)

layout.addWidget(self.summa\_ievade)

layout.addWidget(self.datums\_label)

layout.addWidget(self.datums\_ievade)

layout.addWidget(self.kategorija\_label)

layout.addWidget(self.kategorija\_ievade)

layout.addWidget(self.pievienot\_poga)

self.setLayout(layout)

def pievienot\_kategoriju(self):

jaunais\_nosaukums = self.jauna\_kategorija\_ievade.text().strip()

if not jaunais\_nosaukums:

QMessageBox.warning(self, "Kļūda", "Kategorijas nosaukums nevar būt tukšs!")

return

# Pārbauda, vai kategorija jau eksistē

ekzistejosa = sesija.query(Kategorija).filter\_by(nosaukums=jaunais\_nosaukums, veids="izdevumi").first()

if ekzistejosa:

QMessageBox.warning(self, "Kļūda", "Šāda kategorija jau pastāv!")

return

# Izveido jaunu kategoriju un saglabāt datubāzē

jauna\_kategorija = Kategorija(nosaukums=jaunais\_nosaukums, veids="izdevumi")

sesija.add(jauna\_kategorija)

sesija.commit()

# Atjaunina kategoriju izvēlni

self.ieladet\_kategorijas()

# Notīra ievades lauku

self.jauna\_kategorija\_ievade.clear()

def ieladet\_kategorijas(self):

self.kategorija\_ievade.clear() # Notīra esošās opcijas

kategorijas = sesija.query(Kategorija).filter\_by(veids="izdevumi").all()

for kat in kategorijas:

self.kategorija\_ievade.addItem(kat.nosaukums, kat.id)

def saglabat\_izdevumu(self):

try:

summa = float(self.summa\_ievade.text())

datums = self.datums\_ievade.date().toPyDate()

kategorija\_id = self.kategorija\_ievade.currentData()

if summa <= 0:

raise ValueError("Summai jābūt pozitīvai!")

jauns\_izdevums = Izdevumi(lietotaja\_id=self.lietotaja\_id, summa=summa, datums=datums, kategorija\_id=kategorija\_id)

sesija.add(jauns\_izdevums)

sesija.commit()

self.close()

except ValueError as e:

QMessageBox.warning(self, "Kļūda", str(e))

Parskata\_logs.py

from PyQt5.QtWidgets import QDialog, QLabel, QPushButton, QVBoxLayout, QHBoxLayout, QWidget, QCheckBox, QScrollArea

from dati.datubaze import Ienakumi, Izdevumi, Kategorija, sesija

from PyQt5.QtCore import Qt

from dizains.stils import get\_stils

from functools import partial

class ParskataLogs(QDialog):

def \_\_init\_\_(self, lietotaja\_id, vecais\_logs):

super().\_\_init\_\_()

self.lietotaja\_id = lietotaja\_id

self.vecais\_logs = vecais\_logs

self.setWindowTitle("Budžeta pārskats")

self.setWindowFlags(self.windowFlags() & ~Qt.WindowContextHelpButtonHint) # Noņemt jautājuma zīmi loga stūrī

self.setGeometry(150, 150, 400, 500)

self.setStyleSheet(get\_stils())

self.atzimetie\_ieraksti = []

self.filtrs = "visi"

self.init\_ui()

def init\_ui(self):

self.layout = QVBoxLayout()

self.layout.addWidget(QLabel("Budžeta pārskats", alignment=Qt.AlignCenter))

self.filtru\_rinda = QHBoxLayout()

self.ienakumi\_poga = QPushButton("Ienākumi")

self.ienakumi\_poga.clicked.connect(lambda: self.filtrēt("ienakumi"))

self.izdevumi\_poga = QPushButton("Izdevumi")

self.izdevumi\_poga.clicked.connect(lambda: self.filtrēt("izdevumi"))

self.visi\_poga = QPushButton("Visi")

self.visi\_poga.clicked.connect(lambda: self.filtrēt("visi"))

self.filtru\_rinda.addWidget(self.ienakumi\_poga)

self.filtru\_rinda.addWidget(self.izdevumi\_poga)

self.filtru\_rinda.addWidget(self.visi\_poga)

self.layout.addLayout(self.filtru\_rinda)

self.scroll\_area = QScrollArea()

self.scroll\_area.setWidgetResizable(True)

self.scroll\_widget = QWidget()

self.scroll\_layout = QVBoxLayout(self.scroll\_widget)

self.scroll\_area.setWidget(self.scroll\_widget)

self.layout.addWidget(self.scroll\_area)

self.dzesanas\_poga = QPushButton("Dzēst atzīmētos")

self.dzesanas\_poga.clicked.connect(self.dzest\_atzimetos)

self.layout.addWidget(self.dzesanas\_poga)

self.atpakal\_poga = QPushButton("Atpakaļ")

self.atpakal\_poga.clicked.connect(self.atgriezties)

self.layout.addWidget(self.atpakal\_poga)

self.setLayout(self.layout)

self.paradit\_parskatu()

# Parāda ienākumus un izdevumus uz ekrāna

def paradit\_parskatu(self):

ienakumi = sesija.query(Ienakumi).filter\_by(lietotaja\_id=self.lietotaja\_id).order\_by(Ienakumi.datums.desc()).all() # Sakārto pēc datuma dilstoši

izdevumi = sesija.query(Izdevumi).filter\_by(lietotaja\_id=self.lietotaja\_id).order\_by(Izdevumi.datums.desc()).all()

for i in reversed(range(self.scroll\_layout.count())):

self.scroll\_layout.itemAt(i).widget().deleteLater() # Notīra visus iepriekšējos ierakstus

if self.filtrs == "visi" or self.filtrs == "ienakumi": # Parāda tikai ienākumus, ja filtrs ir ienākumi

self.pievienot\_ierakstus("IENĀKUMI:", ienakumi)

if self.filtrs == "visi" or self.filtrs == "izdevumi": # Parāda tikai izdevumus, ja filtrs ir izdevumi

self.pievienot\_ierakstus("IZDEVUMI:", izdevumi)

# Pievieno ierakstus uz ekrāna

def pievienot\_ierakstus(self, virsraksts, ieraksti):

self.scroll\_layout.addWidget(QLabel(virsraksts))

for ieraksts in ieraksti:

ieraksta\_widget = QWidget()

ieraksta\_layout = QHBoxLayout()

checkbox = QCheckBox()

checkbox.stateChanged.connect(partial(self.atzimet\_ierakstu, ieraksts))

# Iegūt kategorijas nosaukumu no saistītās tabulas

kategorija = sesija.query(Kategorija).filter\_by(id=ieraksts.kategorija\_id).first()

kategorijas\_nosaukums = kategorija.nosaukums if kategorija else "Nezināma kategorija"

etikete = QLabel(f"{ieraksts.datums} - {ieraksts.summa} EUR ({kategorijas\_nosaukums})")

ieraksta\_layout.addWidget(checkbox)

ieraksta\_layout.addWidget(etikete)

ieraksta\_widget.setLayout(ieraksta\_layout)

self.scroll\_layout.addWidget(ieraksta\_widget)

# Filtrē ienākumus un izdevumus

def filtrēt(self, tips):

self.filtrs = tips

self.paradit\_parskatu()

# Atzīmē ierakstu, lai to varētu dzēst

def atzimet\_ierakstu(self, ieraksts):

if ieraksts in self.atzimetie\_ieraksti:

self.atzimetie\_ieraksti.remove(ieraksts)

else:

self.atzimetie\_ieraksti.append(ieraksts)

# Dzēš atzīmētos ierakstus

def dzest\_atzimetos(self):

for ieraksts in self.atzimetie\_ieraksti:

sesija.delete(ieraksts)

sesija.commit()

self.atzimetie\_ieraksti.clear() # Tīrām sarakstu

self.paradit\_parskatu()

# Aizver logu un atgriežas pie galvenā loga

def atgriezties(self):

self.close()

if not self.vecais\_logs.isVisible():

self.vecais\_logs.show()

Pieslegsanas\_logs.py

from PyQt5.QtWidgets import QDialog, QLabel, QLineEdit, QPushButton, QVBoxLayout, QMessageBox

from dati.datubaze import Lietotajs, sesija

from dizains.stils import get\_stils

from PyQt5.QtCore import Qt

class PieslegsanasLogs(QDialog):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Pieslēgšanās")

self.setWindowFlags(self.windowFlags() & ~Qt.WindowContextHelpButtonHint)

self.setGeometry(100, 100, 300, 200)

self.setStyleSheet(get\_stils())

self.init\_ui()

def init\_ui(self):

layout = QVBoxLayout()

self.lietotajvards\_label = QLabel("Lietotājvārds:")

self.lietotajvards\_ievade = QLineEdit()

self.parole\_label = QLabel("Parole:")

self.parole\_ievade = QLineEdit()

self.parole\_ievade.setEchoMode(QLineEdit.Password)

self.pieslegties\_poga = QPushButton("Pieslēgties")

self.pieslegties\_poga.clicked.connect(self.pieslegties)

self.registreties\_poga = QPushButton("Reģistrēties")

self.registreties\_poga.clicked.connect(self.registreties)

layout.addWidget(self.lietotajvards\_label)

layout.addWidget(self.lietotajvards\_ievade)

layout.addWidget(self.parole\_label)

layout.addWidget(self.parole\_ievade)

layout.addWidget(self.pieslegties\_poga)

layout.addWidget(self.registreties\_poga)

self.setLayout(layout)

# Funkcija, kas tiek izsaukta, kad lietotājs nospiež "Pieslēgties" pogu

def pieslegties(self):

lietotajvards = self.lietotajvards\_ievade.text().strip() # Iegūstam ievadīto lietotājvārdu

parole = self.parole\_ievade.text().strip() # Iegūstam ievadīto paroli

if not lietotajvards or not parole:

QMessageBox.warning(self, "Kļūda", "Lūdzu aizpildiet visus laukus!")

return

lietotajs = sesija.query(Lietotajs).filter\_by(lietotajvards=lietotajvards).first() # Meklē lietotāju pēc lietotājvārda

if lietotajs and lietotajs.parbaud\_paroli(parole): # Ja lietotājs eksistē un parole ir pareiza

self.lietotaja\_id = lietotajs.id # Saglabājam lietotāja ID

self.accept() # Aizver pieslēgšanās logu

else:

QMessageBox.warning(self, "Kļūda", "Nepareizs lietotājvārds vai parole!")

# Funkcija, kas tiek izsaukta, kad lietotājs nospiež "Reģistrēties" pogu

def registreties(self):

lietotajvards = self.lietotajvards\_ievade.text().strip() # Iegūstam ievadīto lietotājvārdu

parole = self.parole\_ievade.text().strip() # Iegūstam ievadīto paroli

if not lietotajvards or not parole:

QMessageBox.warning(self, "Kļūda", "Lūdzu aizpildiet visus laukus!")

return

if sesija.query(Lietotajs).filter\_by(lietotajvards=lietotajvards).first(): # Pārbauda, vai lietotājvārds jau eksistē

QMessageBox.warning(self, "Kļūda", "Lietotājvārds jau eksistē!") # Ja eksistē, izvada kļūdu

return

jauns\_lietotajs = Lietotajs(lietotajvards=lietotajvards) # Izveido jaunu lietotāju

jauns\_lietotajs.uzstadit\_paroli(parole) # Uzstāda lietotājam paroli

sesija.add(jauns\_lietotajs) # Pievieno jauno lietotāju datubāzei

sesija.commit() # Saglabā izmaiņas datubāzē

self.lietotaja\_id = jauns\_lietotajs.id # Saglabā jaunā lietotāja ID

self.accept() # Aizver pieslēgšanās logu

Palaidejs.py

import sys

from PyQt5.QtWidgets import QApplication

from logi.galvenais\_logs import BudzetaAplikacija

from logi.pieslegsanas\_logs import PieslegsanasLogs

from PyQt5.QtWidgets import QDialog

if \_\_name\_\_ == "\_\_main\_\_": # Ja šis fails tiek palaists kā galvenais

app = QApplication(sys.argv) # Izveido aplikācijas instanci

pieslegsanas\_logs = PieslegsanasLogs() # Izveido pieslēgšanās loga instanci

if pieslegsanas\_logs.exec\_() == QDialog.Accepted: # Ja lietotājs pieslēdzās

logs = BudzetaAplikacija(pieslegsanas\_logs.lietotaja\_id) # Izveido galvenā loga instanci

logs.show() # Parāda galveno logu

sys.exit(app.exec\_()) # Palaiž aplikāciju