

Rīgas Tālmācības vidusskola

BUDŽETA APLIKĀCIJAS DOKUMENTĀCIJA

Piekļuves darbs programmēšanā

Darba autors: Raimonds Zlotņikovs

(2025)

Satura rādītājs

1. Problēmas izpēte
 - 1.1. Procesa izvēle un pamatojums
 - 1.2. Izpētes procesa apraksts
 - 1.3. Izpētes datu apkopojums
2. Aplikācijas prasības un funkcionalitāte
 - 2.1. Mērķauditorija un tās raksturojums
 - 2.2. Programmatūras un tās funkcijas apraksts
 - 2.3. Plānotās funkcijas
 - 2.4. Datu bāzes projektējums
 - 2.5. Saskarņu īpašības
 - 2.6. Saskarnes karkasa diagramma un skice
 - 2.7. Dizains
 - 2.8. Izvēlēta valoda un vide
3. Izstrādes plāns
 - 3.1. Izstrādes modelis
 - 3.2. Izstrādes posmi
 - 3.3. Testēšana, dokumentēšana un izlaišana
 - 3.4. Uzturēšana un uzlabojumi
4. Atklūdošanas un akceptēšanas pārskats
 - 4.1. Atklūdošanas process
 - 4.1.1. Metodoloģija
 - 4.1.2. Galvenās problēmas un to novēršana
 - 4.1.2.1. Nepareiza bilances aprēķināšana un noapaļošana
 - 4.1.2.2. Kategoriju manuāla ievade
 - 4.1.2.3. Nepareiza datuma izvēle
 - 4.1.2.4. Balance netika atjaunota pēc ierakstu dzēšanas
 - 4.1.2.5. Nepareiza lietotāja sesijas pārvaldība
 - 4.2. Akceptēšanas kritēriji
 - 4.3. Testēšanas rezultāti
 - 4.4. Lietotāju atsauksmes un pēdējie labojumi
 - 4.5. Akceptēšanas lēmums

- 5. Lietotāja ceļvedis
 - 5.1. Lietotāja konta izveide
 - 5.2. Lietotāju autentifikācija
 - 5.3. Ienākumu, izdevumu pievienošana un kategorijas izveide
 - 5.4. Pārskata apskatīšana
 - 5.5. Iziešana no konta
 - 5.6. Aplikācija aizvēršana
- 6. Pielikumi

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 izpēti izriet no aplikācijas lietotājiem, tā kā paredzētais lietotāju skaits ir tikai divi cilvēki kur viens no tiem ir es, nav iespējams 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. Izpēti 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 aplikācijā kuras nepieciešams ieviest;
- **Automatizācijas iespēju noteikšana:** noteikt kurus procesus ir iespējams automatizēt, spriežot pēc novērotajiem procesiem.

1.3. Izpēti datu apkopojums:

- **Lietotāja sūdzības:**
 - a) google konta nepieciešamība - kas rada nesaistītu datu automātisko sinhronizāciju starp pievienotajām ierīcēm, pat pēc šīs sinhronizācijas izlēģšanas, piemēram telefona numuri kur tie tiek pārkopēti no vienas ierīces otrā un sakritīgo nosaukumi tiek mainīti radot apgrūtinājumu atpazīt pareizos kontaktus;
 - b) datu zudumi un nepareiza saglabāšana - bieža reālā budžeta nesakritība ar to kas ir fiksēts aplikācijā;
 - c) reklāmas un maksas funkcijas;
 - d) lietotāja datu saglabāšana un nodošana trešajām pusēm.
- **Lietotāja vajadzības:**
 - a) konta izveide bez trešās puses iesaistes;
 - b) bez reklāmām un bezmaksas;
 - c) dati tiek uzglabāti tikai personīgajās ierīcēs;
 - d) dati tiek uzglabāti un apstrādāti pareizi;
 - e) paroles tiek uzglabātas drošā veidā;
 - f) ienākumu un Izdevumu fiksēšana (datums, summa, kategorija);
 - g) var redzēt visus Ienākumus un Izdevumus;
 - h) jaunu kategoriju izveide;

- i) iespēja izdzēst ierakstus;
- j) iespēja šķirot datus pēc Ienākumiem un Izdevumiem;
- k) redzama kopējā balance;
- l) visi dati tiek šifrēti.

- **Automatizācijas iespējas:**

- a) sinhronizācija starp ierīcēm;
- b) datu pārvešana no citām aplikācijām;
- c) automātiska transakciju nolasīšana no bankas kontiem un pievienošana aplikācijā;
- d) 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 aplikācijai 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 apraksts

Funkcionālās (jābūt iekļautām):

A. Lietotāja reģistrācija un autentifikācija:

- a) reģistrācija bez trešo pušu iesaistes;
- b) droša autentifikācija ar šifrētu paroli;
- c) var izveidot vairākus lietotāju kontus.

B. Datu sinhronizācija un drošība:

- a) dati saglabāti lokāli ierīcēs;
- b) nav reklāmu un maksas funkciju.

C. Budžeta pārvaldība:

- a) var pievienot jaunus Ienākumus un Izdevumus, kuriem klāt jābūt ir datumam, summai un kategorijai;
- b) spēja izdzēst ierakstus.

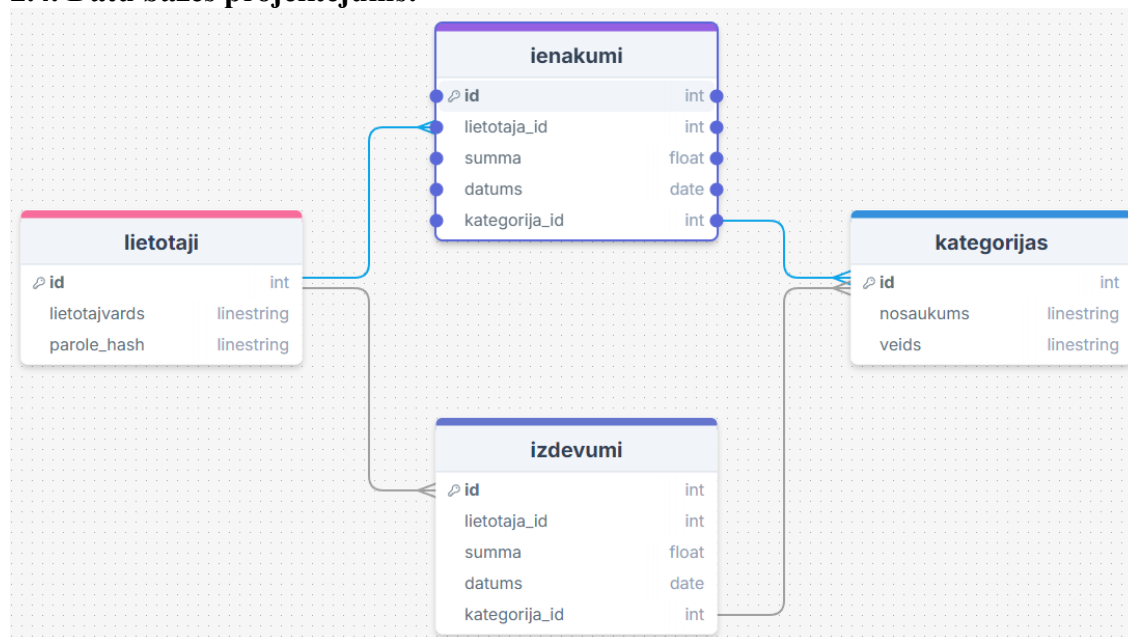
D. Pārskati un analīze:

- a) ienākumu un izdevumu pārskatu var uzrādīt atsevišķi vai kopēji;
- b) ieraksti tiek sagrupēti pēc datumiem dilstošā secībā.

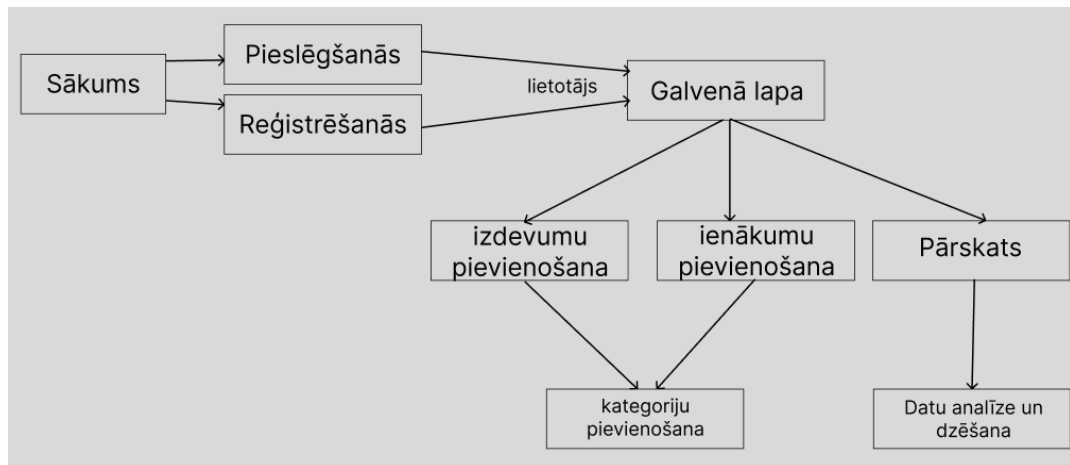
2.3. Plānotās funkcijas:

- 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 ikonām;
- 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 izvēlēta 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).

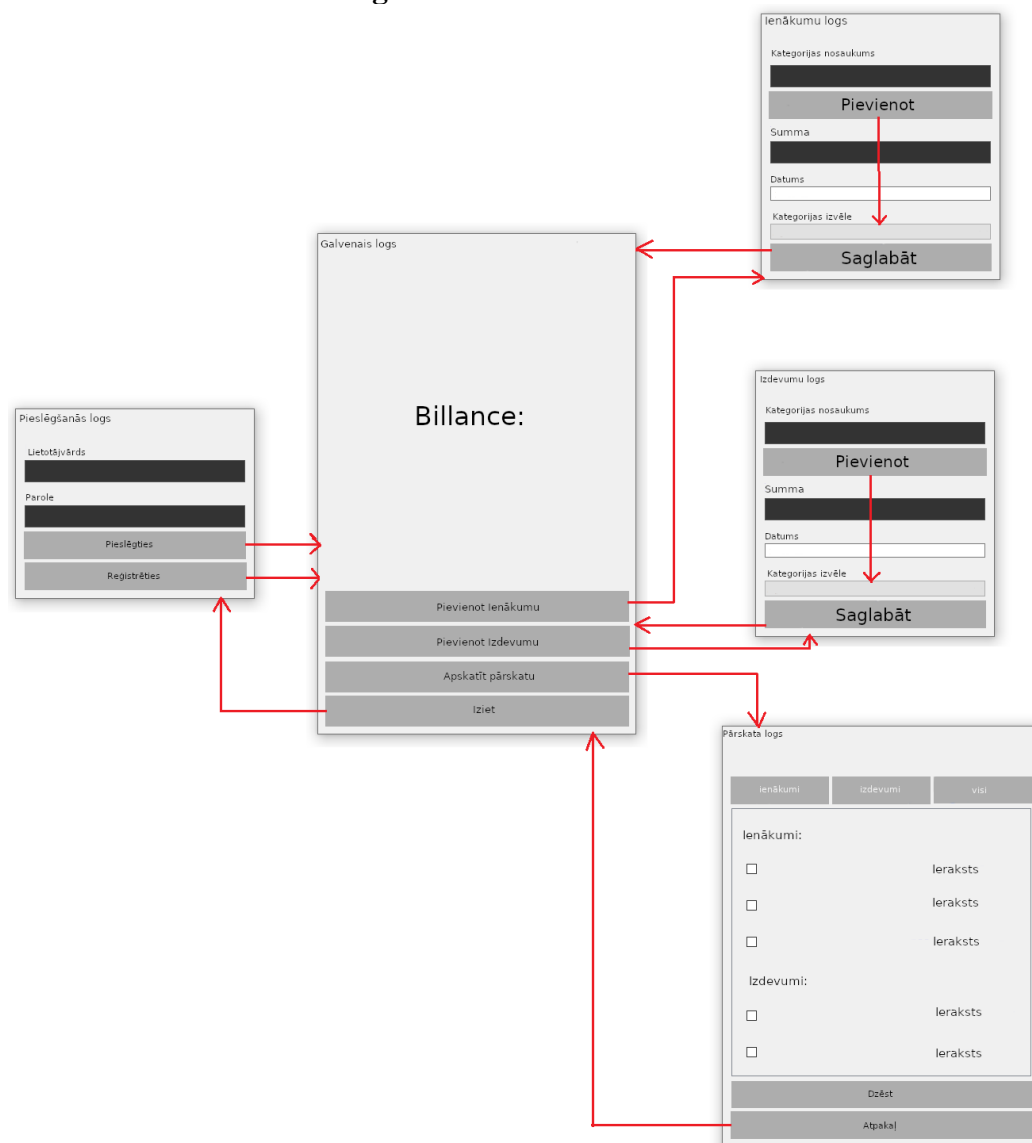
2.4. Datu bāzes projektējums:



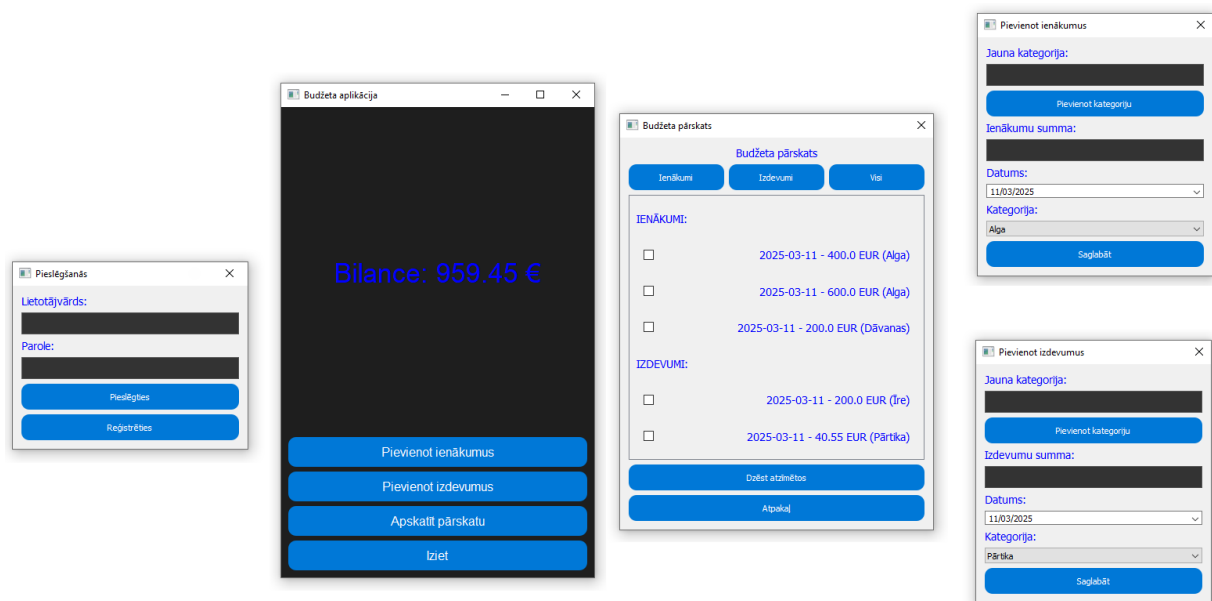
2.5. Saskaņņu īpašības:



2.6. Skares 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:

- SQL (datu bāzes pārvaldībai).
- Bash (automatizācijai).
- JSON (datu apmaiņai).
- PyQt5 (grafiskajai saskarsnei)
- SQLAlchemy (rīks lai atvieglotu darbu ar datubāzi)
- Bcrypt (parolu šifrēšanai)
- 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. Viegļāk un ātrāk pārbaudīt, un veikt labojumus vai papildinājumus pēc funkciju testēšanas.

3.2. Izstrādes posmi:

1. posms: Pamatfunkcionalitātes izstrāde

- Lietotāja reģistrācija un autentifikācija:
 - Izveidot reģistrācijas un autentifikācijas sistēmu.
 - Implementēt drošu parolu šifrēšanu, izmantojot Bcrypt.
- Datu bāzes izveide: izveidot SQLite datu bāzi ar tabulām lietotājiem, ienākumiem, izdevumiem un kategorijām.

- C. Budžeta pārvaldība: ieviest iespēju pievienot, dzēst un apskatīt ienākumus un izdevumus.
- D. Dizaina izstrāde: izveidot kopīgu dizainu aplikācijai.

2. posms: Pārskatu un analīzes funkcionalitāte

- A. Pārskatu izveide:
 - a) Ieviest iespēju apskatīt ienākumus un izdevumus atsevišķi vai kopēji.
 - b) Ierakstu grupēšana pēc datuma dilstošā secībā.

3. posms: Nefunkcionālo prasību ieviešana

- A. Datu drošība: ieviest pilnu datu šifrēšanu.
- B. Papildu funkcionalitātes:
 - a) Ierakstu rediģēšanas un kopēšanas funkcionalitāte;
 - b) Komentāru pievienošana pie ierakstiem;
 - c) Procentuālā vizualizācija;
 - d) Pielāgojamas kategorijas ar ikonām;
 - e) Valūtas izvēle;
 - f) Ierakstu šķirošana;
 - g) Kalendāra ieviešana;
 - h) Atgādinājumi par maksājumiem;
 - i) Ierakstu atzīmēšanas funkcija;
 - j) Vairāku budžeti lietotāja pārvaldība;
 - k) Bankas konta piesaiste un automātiska transakciju ievade;
 - l) Automātiskie maksājumi;
 - m) Mašīnmācīšanās integrācija datu grupēšanai;
 - n) Android aplikācijas izveide.

3.3. Testēšana, dokumentēšana un izlaišana

- A. Testēšana:
 - a) Veikt funkcionālos un drošības testus katrā izstrādes posmā;
 - b) Lietotāju testēšana ar diviem galvenajiem lietotājiem.
- B. Dokumentēšana: programmas dokumentācijas izveide.
- C. Izlaišana:
 - a) Sagatavot un publicēt pirmo stabilo versiju;
 - b) Nodrošināt lietotāja rokasgrāmatu;
 - c) 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. Atklūdošanas un akceptēšanas pārskats

4.1. Atklūdošanas process

4.1.1. Metodoloģija

Atklūdošanas 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.

4.1.2. Galvenās problēmas un to novēršana

4.1.2.1. Nepareiza bilances aprēķināšana un noapaļošana

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

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

4.1.2.2. 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.

4.1.2.3. 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.

4.1.2.4. Balance netika atjaunota pēc ierakstu dzēšanas

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

Risinājums: Pievienots `self.balance_label.setText(self.lasi_bilanci())`, kas nodrošina bilances automātisku atjaunināšanu pēc ierakstu izmaiņām.

4.1.2.5. Nepareiza lietotāja sesijas pārvaldība

Problēma: Pēc izešanas 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.

4.2. Akceptēšanas kritēriji

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

- Balance 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 balance 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).

4.3. 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

4.4. 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.

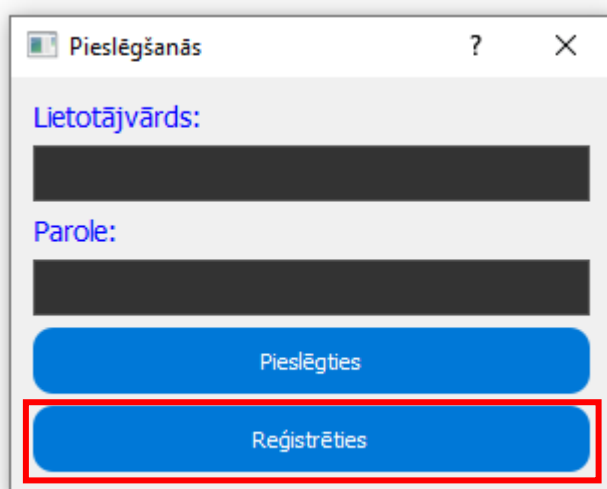
4.5. Akceptēšanas lēmums

Pēc atklūdošanas 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

5.1. Lietotāju konta izveide:

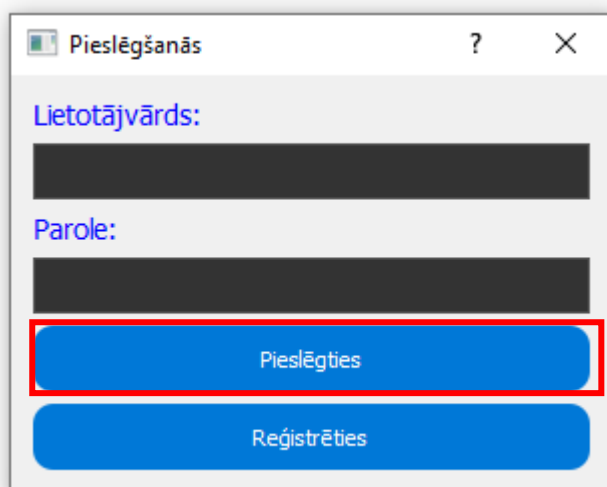
Visas sistēmas funkcijas ir pieejamas katram lietotājam kurš ir reģistrējies. Reģistrācijai ir nepieciešams unikāls lietotājvārds un parole, juri jāievada tiem norādītajos ievades laukos. Pēc lietotājvārda un paroles ievades jānospiež poga “Reģistrēties” kas saglabās ievadītos datus datubāzē un automātiski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



The screenshot shows a window titled "Pieslēgšanās" (Login). It contains two input fields: "Lietotājvārds:" (Username) and "Parole:" (Password). Below these fields are two blue buttons: "Pieslēgties" (Login) and "Reģistrēties" (Register). The "Reģistrēties" button is highlighted with a red rectangular border.

5.2. Lietotāju autentificēšanās:

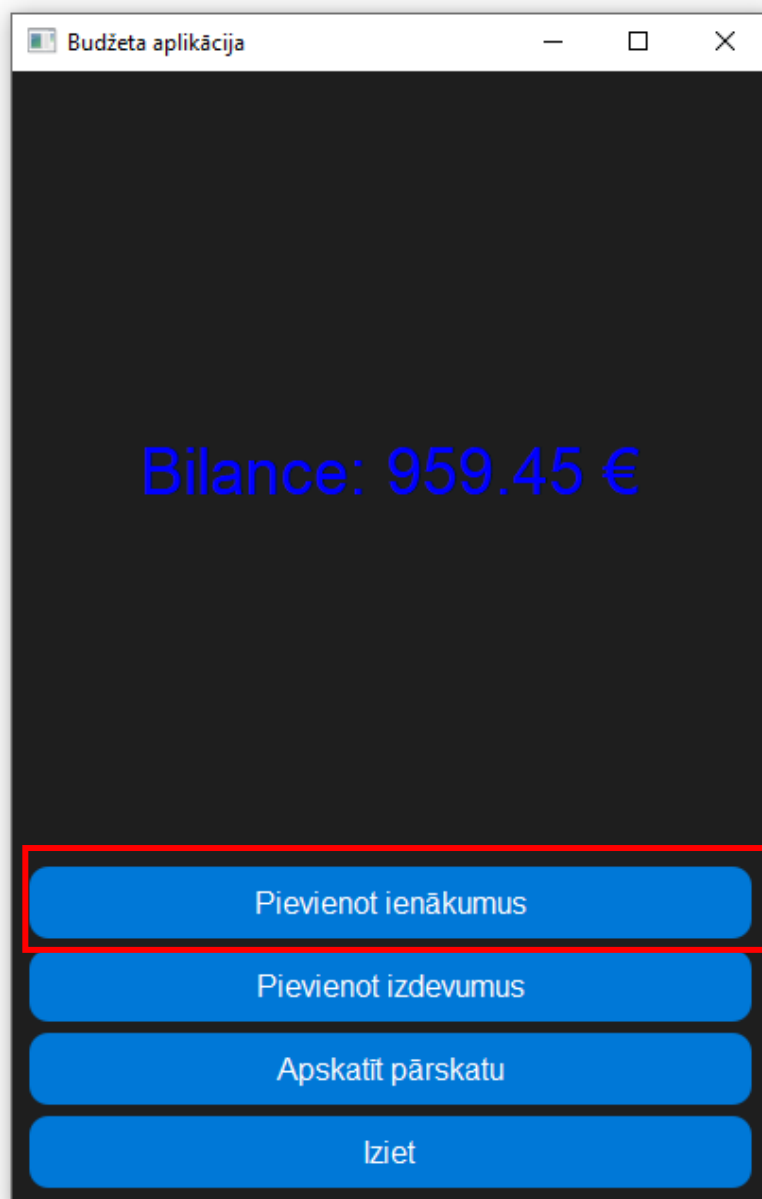
Ja lietotājs jau ir reģistrējies, viņam ir nepieciešams ievadīt lietotājvārdu un paroli ar kuriem ir reģistrēts aplikācija, tiem paredzētajos laukos un nospiež pogu “Pieslēgties” kas pārbaudīs vai lietotājs ir reģistrēts un pēc apstiprināšanas novirzīs to uz galveno lapu.



The screenshot shows the same "Pieslēgšanās" window. In this view, the "Pieslēgties" (Login) button is highlighted with a red rectangular border, while the "Reģistrēties" button is no longer highlighted.

5.3. Ienākumu, izdevumu pievienošana un kategorijas izveide:

- Lai varētu ievadīt jaunu ienākumu vai izdevumu, galvenajā lapā ir nepieciešams nospiegt pogu “Pievienot ienākumu” vai “Pievienot izdevumu” kas atvērs attiecīgos logus datu ievadei.
- Pirms ienākuma vai izdevumu pievienošanas, sākumā jāpievieno kategoriju, lai to izdarītu nepieciešams ievadīt kategorijas nosaukumu ievades laukā zem “Jauna kategorija” un nospiegt pogu “Pievienot kategoriju”
- Pēc tam 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 nospiegt simbolu x kas atrodas loga augšējā labējā stūrī.



Pievienot ienākumus

Jauna kategorija:

Pievienot kategoriju

Ienākumu summa:

Datums:

11/03/2025

Kategorija:

Alga

Saglabāt

Pievienot izdevumus

Jauna kategorija:

Pievienot kategoriju

Izdevumu summa:

Datums:

11/03/2025

Kategorija:

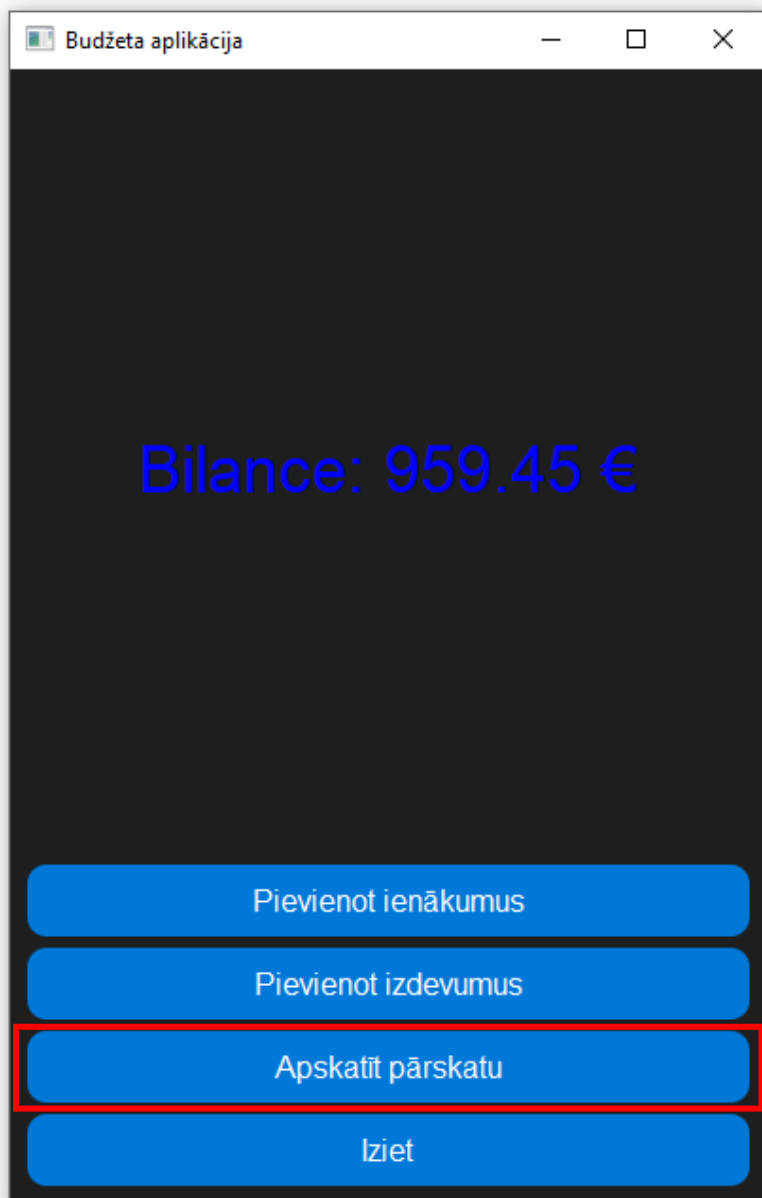
Pārtika

Saglabāt

5.4. 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 atlasīšanu. Nospiežot šīs pogas pārskatā tiek parādīti tikai ienākumi, tikai izdevumi vai visi ieraksti.

- Nospiežot un atļeksējot 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.



Budžeta pārskats

Budžeta pārskats

Ienākumi

Izdevumi

Visi

IENĀKUMI:

☐

2025-03-11 - 400.0 EUR (Alga)

☐

2025-03-11 - 600.0 EUR (Alga)

☐

2025-03-11 - 200.0 EUR (Dāvanas)

IZDEVUMI:

☐

2025-03-11 - 200.0 EUR (Īre)

☐

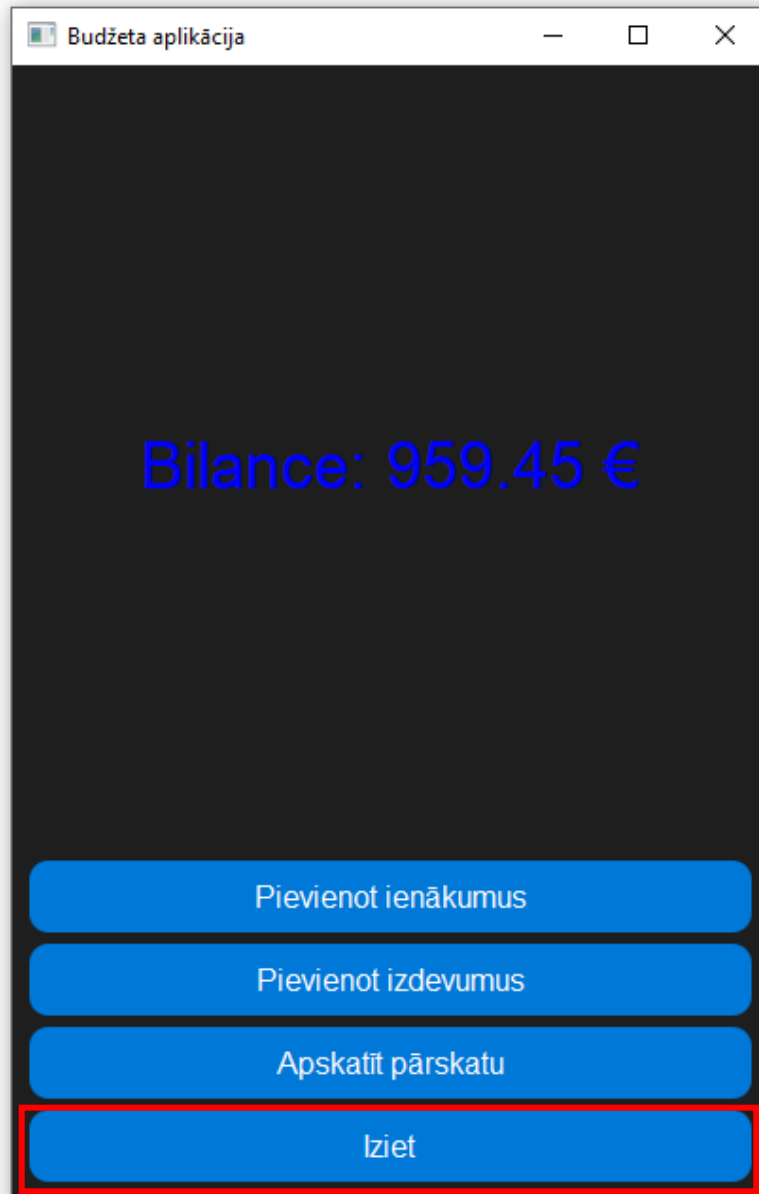
2025-03-11 - 40.55 EUR (Pārtika)

Dzēst atzīmētos

Atpakaļ

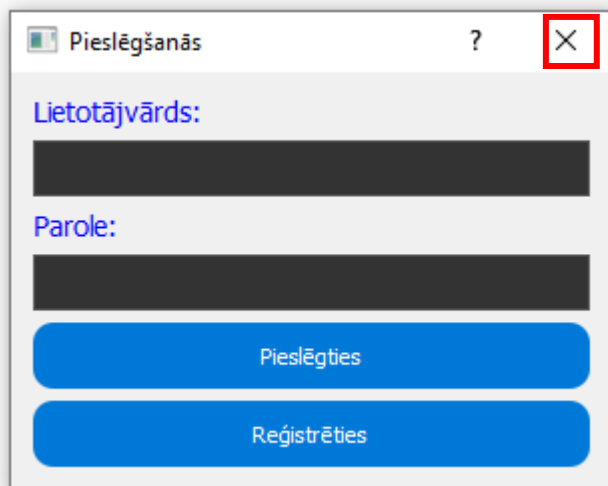
5.5. Iziešana no konta:

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.



5.6. 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.



Balance.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
    balance = ienakumu_summa - izdevumu_summa # Aprēķina bilanci (ienākumi - izdevumi)

    return f"Ienākumi: {ienakumu_summa:.2f}, Izdevumi: {izdevumu_summa:.2f}, Balance:
    {balance:.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
base = declarative_base()

# Lietotāju tabula
class Lietotajs(base):
    __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 uzstadiť_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
kontaineris = QWidget()                                # Izveido konteineru
kontaineris.setLayout(layout)                          # Piešķir konteineram izkārtojumu
self.setCentralWidget(kontaineris)                    # 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.balance_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.balance_label.setText(self.lasi_bilanci())      # Atjauno bilances tekstu

# Atgriež bilances vērtību
def lasi_bilanci(self):                                # Izveido funkciju lasi_bilanci
    dati_balance = generet_parskatu(self.lietotaja_id) # Iegūst bilances datus pēc pieslēgušos
lietotāja
    return f"{dati_balance.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.balance_label.setText(self.lasi_bilanci())    # Atjauno bilances tekstu

```

```

def iziet_no_konta(self):      # Izveido funkciju iziet_no_konta
    self.close()              # Aizver pašreizējo galveno logu

    pieslegšanas_logs = PieslegšanasLogs()      # Izveido pieslēgšanās logu
    if pieslegšanas_logs.exec_() == QDialog.Accepted: # Ja lietotājs veiksmīgi pieslēdzas
        self.__init__(pieslegšanas_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
    pieslegšanas_logs = PieslegšanasLogs()      # Izveido pieslēgšanās logu

    if pieslegšanas_logs.exec_() == QDialog.Accepted:      # Ja lietotājs veiksmīgi
        pieslēdzas
        lietotaja_id = pieslegšanas_logs.lietotaja_id      # Saglabā lietotāja id
        galvenais_logs = BudzetaAplikacija(lietotaja_id, pieslegšanas_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

```

```

    ekzistēja = sesija.query(Kategorija).filter_by(nosaukums=jaunais_nosaukums,
veids="izdevumi").first()
    if ekzistēja:
        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) #
        Noņemti 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()
```

Pieslegšanas_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 PieslegšanasLogs(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.paraud_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.uzstadi_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.pieslegšanas_logs import PieslegšanasLogs
from PyQt5.QtWidgets import QDialog

if __name__ == "__main__":
    app = QApplication(sys.argv)                                # Ja šis fails tiek palaists kā galvenais
                                                            # Izveido aplikācijas instanci

    pieslegšanas_logs = PieslegšanasLogs()                    # Izveido pieslēgšanās loga instanci
    if pieslegšanas_logs.exec_() == QDialog.Accepted:          # Ja lietotājs pieslēdzās
        logs = BudzetaAplikacija(pieslegšanas_logs.lietotaja_id) # Izveido galvenā loga instanci
        logs.show()                                            # Parāda galveno logu
        sys.exit(app.exec_())                                  # Palaiž aplikāciju

```