PROJEKTNI ZADATAK  
Upravljanje radom banke

Studenti:

1. Ibrahim Okić
2. Asja Hadžić
3. Sabrina Bešlagić

Sadržaj

[1 Tema 1](#_Toc84573880)

[2 Izrada i predavanje projekta 1](#_Toc84573881)

[3 Tehnički zahtjevi 1](#_Toc84573882)

[4 Formalni zahtjevi 2](#_Toc84573883)

[5 Izvještaj o projektu 3](#_Toc84573884)

# Tema

Napisati program za upravljanje radom banke.

# Izrada i predavanje projekta

1. Projekat rade grupe po najviše 4 studenta.
2. Rok za predaju projekta je 20.12.2023. godine.
3. Završen CodeBlocks C++17 projekat i Word dokument arhivirati kao ZIP datoteku i poslati na e-mail adresu predmetnog asistenta sa subject-om “PJIP Projekat”. Obavezno obrisati izvršne/objektne datoteke (.exe i .o) iz projekta prije arhiviranja!
4. Projekat se predaje i ocjenjuje prije izlaska na završni ispit.
5. Ukoliko student odluči ne raditi projekat, može izaći na završni ispit.
6. Ukoliko student pošalje projekat nakon termina završnog ispita, mora ponovo izaći na završni ispit kako bi se ostvarili bodovi na projekat.
7. Ispunjenjem tehničkih i formalnih zahtjeva grupa studenata ostvaruje po 20 bodova.
8. Djelimičnim ispunjenjem tehničkih i formalnih zahtjeva asistent odlučuje o broju bodova koji će se dodjeliti grupi za projekat.

# Tehnički zahtjevi

1. Omogućiti operacije unosa, izmjene, pregleda i brisanja: korisnika, radnika, tekućih i štednih računa, kredita i transakcija.
2. Unos novih stavki, te izmjenu postojećih vršiti sa tastature.
3. Željenu radnju (unos, izmjena, pregled, brisanje) korisnik unosi preko tastature na osnovu ponuđenih opcija u terminalu.
4. Za novi unos definirati potrebne kolone i ograničiti unos ukoliko se ne unesu potrebni podaci.
5. Za povezivanje tabela koristit će se ključevi, koji predstavljaju jedinstveni identifikator određene stavke, kao npr:

* radnik/korisnik: tekstualni zapis JMBG od 13 cifara,
* tekući/štedni račun: tekstualni zapis od 16 cifara;

1. Napraviti dokumentaciju projekta u obliku Word dokumenta. Koristiti ovaj dokument kao šablon, te početi sa 5. poglavljem svoj izvještaj o projektu.

# Formalni zahtjevi

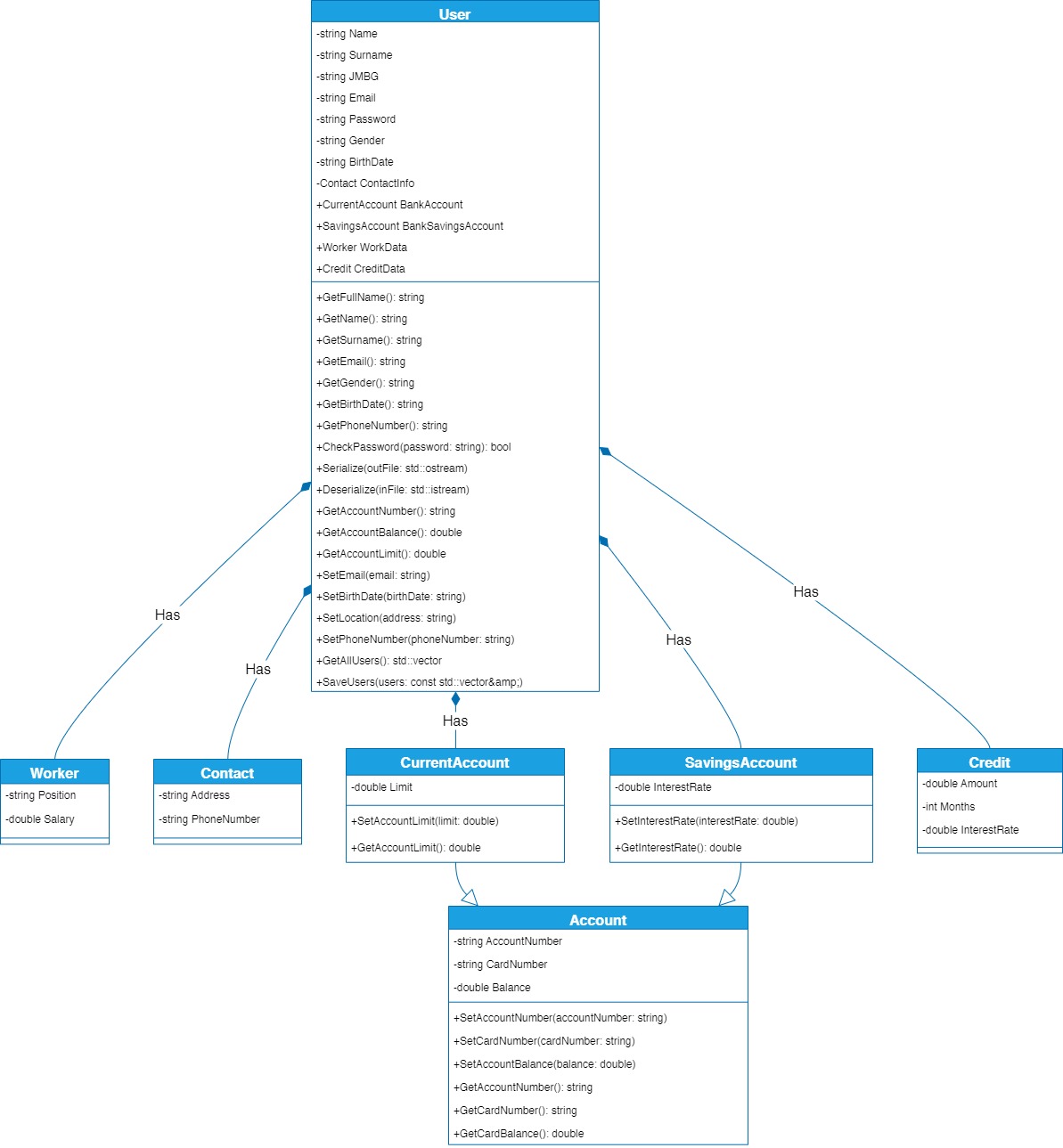
1. Programski kôd (*backend*) pisati u C++ programskom jeziku.
2. Koristiti što više elemenata C++17 standarda.
3. Programsko sučelje (*frontend*) bit će terminal (*Command Prompt* ili UNIX terminal).
4. Strogo se pridržavati principa objektno – orijentiranog programiranja.
5. Jasno definirati potrebne klase.
6. Koristiti naslijeđivanje gdje je potrebno.
7. Maksimalno ograničiti vidljivost i pristup članovima klase.
8. Implementirati sve konstruktore i destruktore.
9. Napraviti odgovarajuće verzije konstruktora ovisno o očekivanom unosu.
10. Koristiti STL kontejnere, a gdje to nije moguće koristiti dinamički alociranu memoriju.
11. Smanjiti ponavljanje kôda kroz implementaciju funkcija.
12. Gdje je moguće koristiti metode umjesto funkcija.
13. Deklaracije izdvojiti u .h datoteke, definicije u .cpp datoteke, a napraviti i main.cpp datoteku koja predstavlja glavni program.
14. Paziti na zauzimanje resursa radne memorije.
15. Objekte, klase, članove klase i njene metode imenovati razumljivo.
16. Nastojati na jedinstven način imenovati metode koji obavljaju iste funkcije u više klasa.

# Izvještaj o projektu

Sama izrada projekta **‘Upravljanje radom banke’** trajala je oko 7 dana gdje sam svaki dan par sati radio na izradi i dizajniranju samih funkcionalnosti ove aplikacije.

Prije nego što sam krenuo pisati sam kod htio sam grafički prikazati kako bi izgledao cijeli rad aplikacije pomoću dijagrama. Za izradu dijagrama koristio sam **draw.io** web stranicu i koristio sam se Klasnim dijagramima i Sekvencijalnim dijagramima.

* Klasni dijagram u koji sadrži sve klase korištene u aplikaciji:



Klasa ***“User”*** predstavlja korisnika unutar bankarskog sistema, dizajniranu za čuvanje i upravljanje informacijama vezanim za lične podatke, kontakt informacije, bankovne račune, podatke o zaposlenju i podatke o kreditu. Klasa obuhvata različite atribute i metode za interakciju s korisničkim podacima.

**Privatni Članovi**

* Name (string): Ime korisnika.
* Surname (string): Prezime korisnika.
* JMBG (string): Jedinstveni identifikacioni broj korisnika.
* Email (string): Email adresa korisnika.
* Password (string): Lozinka povezana s korisničkim računom.
* Gender (string): Pol korisnika.
* BirthDate (string): Datum rođenja korisnika.
* ContactInfo (Contact): Objekat koji sadrži kontakt informacije korisnika.
* BankAccount (CurrentAccount): Informacije o tekućem računu korisnika.
* BankSavingsAccount (SavingsAccount): Informacije o štednom računu korisnika.
* WorkData (Worker): Informacije o zaposlenju korisnika.
* CreditData (Credit): Informacije o eventualnom kreditu povezanom s korisnikom.

**Javni Članovi**

* Konstruktor
* User(IUser props = {}): Konstruiše novi objekat korisnika, inicijalizujući atribute sa dostavljenim svojstvima ili podrazumijevanim vrijednostima.

**Metode Dohvata**

* GetFullName(): string: Dobavlja puno ime korisnika.
* GetName(): string: Dobavlja ime korisnika.
* GetSurname(): string: Dobavlja prezime korisnika.
* GetEmail(): string: Dobavlja email adresu korisnika.
* GetGender(): string: Dobavlja pol korisnika.
* GetBirthDate(): string: Dobavlja datum rođenja korisnika.
* GetPhoneNumber(): string: Dobavlja broj telefona korisnika.
* CheckPassword(string password): bool: Provjerava da li unesena lozinka odgovara lozinci korisnika.
* GetAccountNumber(): string: Dobavlja broj tekućeg računa korisnika.
* GetAccountBalance(): double: Dobavlja saldo tekućeg računa korisnika.
* GetAccountLimit(): double: Dobavlja limit na tekućem računu korisnika.

**Metode Postavljanja**

* SetEmail(string email): Postavlja email adresu korisnika.
* SetBirthDate(string birthDate): Postavlja datum rođenja korisnika.
* SetLocation(string address): Postavlja adresu korisnika.
* SetPhoneNumber(string phoneNumber): Postavlja broj telefona korisnika.

**Statičke Metode**

* static std::vector<User> GetAllUsers(): Dohvata vektor koji sadrži sve korisnike iz datoteke sa podacima.
* static void SaveUsers(const std::vector<User> &users): Sprema vektor korisnika u datoteku sa podacima.

**Metode Serijalizacije**

* void Serialize(std::ostream &outFile) const: Serijalizuje objekat korisnika u izlazni tok.
* void Deserialize(std::istream &inFile): Deserijalizuje objekat korisnika iz ulaznog toka.



Primjer korištenja klase “User”

Klasa ***“Account”***predstavlja opću strukturu računa, sa atributima koji uključuju broj računa, broj kartice i saldo. Ova klasa omogućava postavljanje i dohvaćanje osnovnih informacija o računu.

**Privatni Članovi**

* AccountNumber (string): Broj računa korisnika.
* CardNumber (string): Broj kartice vezane uz račun.
* Balance (double): Saldo na računu.

**Javni Članovi**

* Konstruktor
* Account(): Konstruiše objekat računa postavljajući podrazumijevane vrijednosti.

**Metode Postavljanja**

* SetAccountNumber(string accountNumber): Postavlja broj računa.
* SetCardNumber(string cardNumber): Postavlja broj kartice.
* SetAccountBalance(double balance): Postavlja saldo na računu.

**Metode Dohvata**

* GetAccountNumber() const: string: Dohvaća broj računa.
* GetCardNumber() const: string: Dohvaća broj kartice.
* GetCardBalance() const: double: Dohvaća saldo na računu.

Klasa ***“CurrentAccount”*** naslijeđuje osnovnu strukturu klase Account i dodaje mogućnost postavljanja i dohvaćanja limita na tekućem računu.

**Dodatni Članovi**

* Limit (double): Maksimalni iznos koji korisnik može imati na tekućem računu.
* Javni Članovi
* Konstruktor
* CurrentAccount(): Konstruiše objekat tekućeg računa postavljajući podrazumijevane vrijednosti, uključujući limit.

**Metode Postavljanja**

* SetAccountLimit(double limit): Postavlja limit na tekućem računu.

**Metode Dohvata**

* GetAccountLimit() const: double: Dohvaća limit na tekućem računu.

Klasa ***“SavingsAccount”*** naslijeđuje osnovnu strukturu klase Account i omogućava postavljanje i dohvaćanje kamatne stope na štednom računu.

**Dodatni Članovi**

* InterestRate (double): Kamatna stopa na štednom računu.

**Javni Članovi**

* Konstruktor
* SavingsAccount(): Konstruiše objekat štednog računa postavljajući podrazumijevane vrijednosti, uključujući kamatnu stopu.

**Metode Postavljanja**

* SetInterestRate(double interestRate): Postavlja kamatnu stopu na štednom računu.

**Metode Dohvata**

* GetInterestRate() const: double: Dohvaća kamatnu stopu na štednom računu.

Klasa ***„Credit“***predstavlja strukturu koja omogućava praćenje informacija o kreditu, uključujući iznos, trajanje otplate u mjesecima i kamatnu stopu. Ova klasa omogućava postavljanje i dohvaćanje tih informacija.

**Privatni Članovi**

* **Amount (double):** Iznos kredita.
* **Months (int):** Trajanje otplate kredita u mjesecima.
* **InterestRate (double):** Kamatna stopa kredita.

**Javni Članovi**

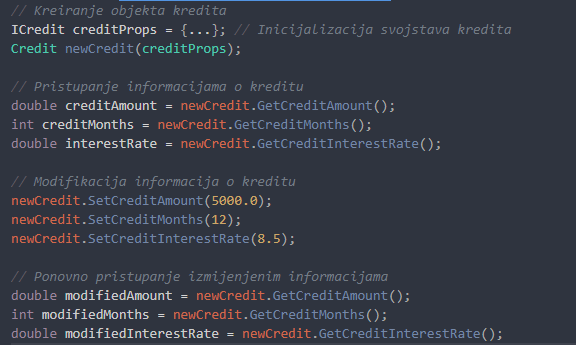
* Konstruktor
* **Credit(ICredit props = {}):** Konstruiše objekat kredita postavljajući svojstva iz proslijeđenog objekta.

**Metode Postavljanja**

* **SetCreditAmount(double amount):** Postavlja iznos kredita.
* **SetCreditMonths(int months):** Postavlja trajanje otplate kredita u mjesecima.
* **SetCreditInterestRate(double interestRate):** Postavlja kamatnu stopu kredita.

**Metode Dohvata**

* **GetCreditAmount() const: double:** Dohvaća iznos kredita.
* **GetCreditMonths() const: int:** Dohvaća trajanje otplate kredita u mjesecima.
* **GetCreditInterestRate() const: double:** Dohvaća kamatnu stopu kredita.



Primjer korištenja klase “Credit”

Klasa ***„Worker“*** predstavlja strukturu koja omogućava praćenje informacija o zaposlenju, uključujući radnu poziciju i platu. Ova klasa omogućava postavljanje i dohvaćanje tih informacija.

**Privatni Članovi**

* **Position (string):** Radna pozicija zaposlenika.
* **Salary (double):** Plata zaposlenika.

**Javni Članovi**

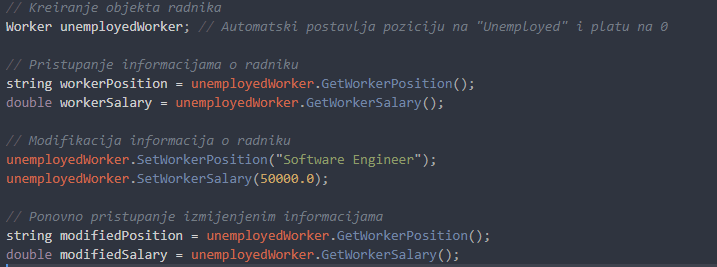
* Konstruktori
* **Worker():** Konstruiše objekat radnika postavljajući podrazumijevane vrijednosti (nije zaposlen, nema plate).
* **Worker(string position, double salary):** Konstruiše objekat radnika postavljajući radnu poziciju i platu na osnovu proslijeđenih vrijednosti.

**Metode Postavljanja**

* **SetWorkerPosition(string position):** Postavlja radnu poziciju zaposlenika.
* **SetWorkerSalary(double salary):** Postavlja platu zaposlenika.

**Metode Dohvata**

* **GetWorkerPosition() const: string:** Dohvaća radnu poziciju zaposlenika.
* **GetWorkerSalary() const: double:** Dohvaća platu zaposlenika.ž



Primjera korištenja klase „Worker“

Klasa ***„Transaction“*** predstavlja strukturu koja omogućava praćenje informacija o transakcijama, uključujući tip transakcije, puna imena primaoca i pošiljaoca, iznos, te informacije o računima sa kojih se vrši transakcija. Ova klasa omogućava i serijalizaciju i deserijalizaciju objekata transakcija, kao i manipulaciju transakcijama putem statičkih metoda.

**Privatni Članovi**

* **Type (string):** Tip transakcije (npr., "Transfer", "Deposit", "Withdrawal").
* **ReceiverFullName (string):** Puno ime primaoca transakcije.
* **SenderFullName (string):** Puno ime pošiljaoca transakcije.
* **Amount (double):** Iznos transakcije.
* **From (string):** Račun sa kojeg se vrši transakcija.
* **To (string):** Račun na koji se vrši transakcija.

**Javni Članovi**

* Konstruktor
* **Transaction(ITransaction props = {}):** Konstruiše objekat transakcije postavljajući svojstva iz proslijeđenog objekta.

**Serijalizacija i Deserijalizacija**

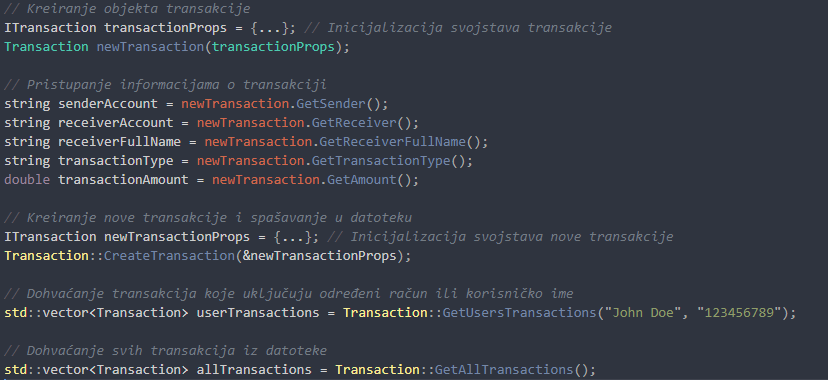
* **void Serialize(std::ostream &outFile) const:** Serijalizuje objekat transakcije u izlazni tok.
* **void Deserialize(std::istream &inFile):** Deserijalizuje objekat transakcije iz ulaznog toka.

**Metode Dohvata**

* **string GetSender():** Dohvaća informacije o računu pošiljaoca transakcije.
* **string GetReceiver():** Dohvaća informacije o računu primaoca transakcije.
* **string GetReceiverFullName():** Dohvaća puno ime primaoca transakcije.
* **string GetTransactionType():** Dohvaća tip transakcije.
* **double GetAmount():** Dohvaća iznos transakcije.

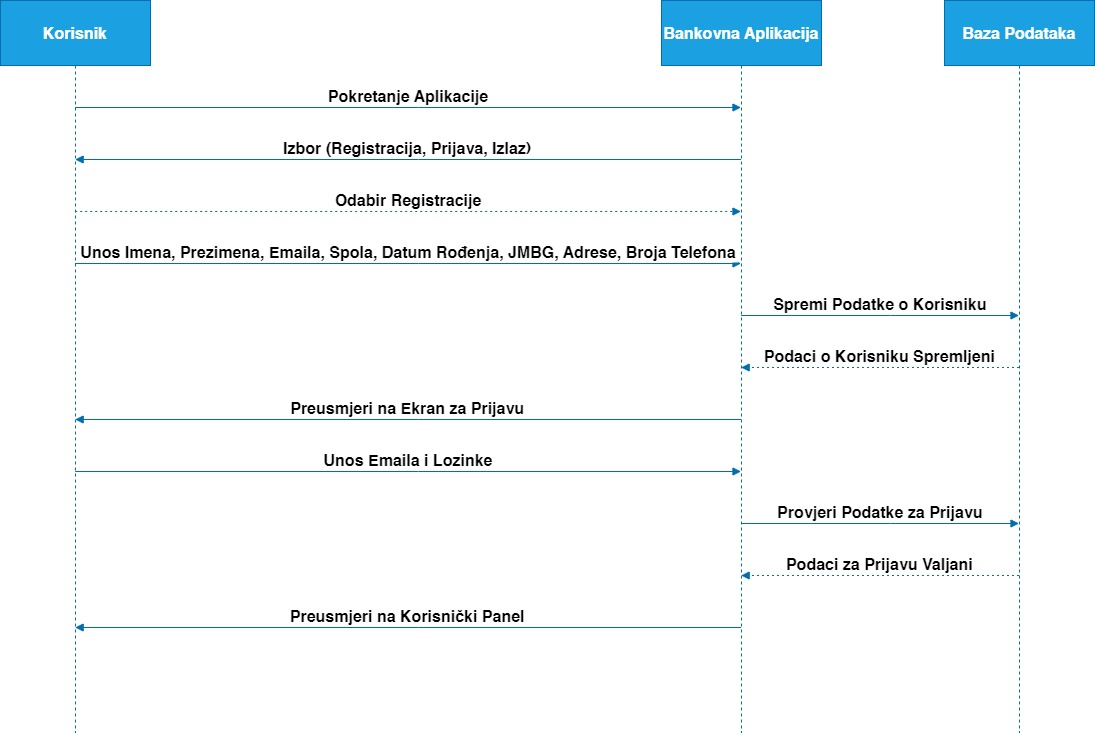
**Statičke Metode**

* \**static void CreateTransaction(ITransaction props):* Kreira novu transakciju na osnovu proslijeđenih svojstava i sprema je u datoteku.
* **static std::vector<Transaction> GetUsersTransactions(string name, string accountNumber):** Dohvaća sve transakcije koje uključuju određeni račun (pošiljaoca ili primaoca) ili korisničko ime.
* **static std::vector<Transaction> GetAllTransactions():** Dohvaća sve transakcije iz datoteke.



Primjer korištenja klase “Transaction”

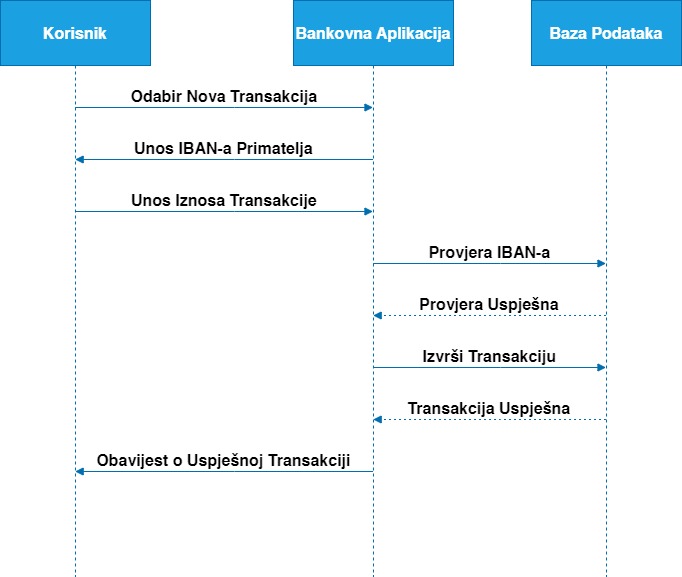
* Proces prijave i registracije na aplikaciju izgleda ovako:



* Opcije koje se nalaze na glavnom korisničkom sučelju koje se pojavljuje nakon prijave:



* Proces transakcije:



* Administrativni dio za zaposlene osobe u banci:

