

ASSIGNMENT-2

STATEMENT:

| | |
|---------------------|--|
| Assignment 2 | * Design a system that replicates a working of a banking system. It will contain multiple banks and branches, a customer can open a savings account or can take a loan at an interest rate of 12%, can view his account, transactions and loan details (such as loan pending, interest to be paid this year etc), a customer can perform various actions such as deposit and withdraw cash, repay loan, take a loan, etc |
| | * List all the features |
| | * Design a database ER diagram |
| | * list all the APIs |
| | * Implement all APIs |

To do this assignment first i need to determine what entities are there in this ->

- **Bank**
- **Branch**
- **Customer**
- **Account**
- **Transaction**
- **Loan**
- **LoanPayment**

Now after identifying these entities I need to see how these are related also from question what i think is specified:

Multiple Banks -----> Need API, Database connection

Multiple Branches -----> Need API, Database connection

Customer opens Savings Account-----> API, Database Connection, Business Logic

Loan Take with interest rate 12% -----> API, Database Connection, Business Logic

Customer can view his account -----> API, Database Connection, Business Logic

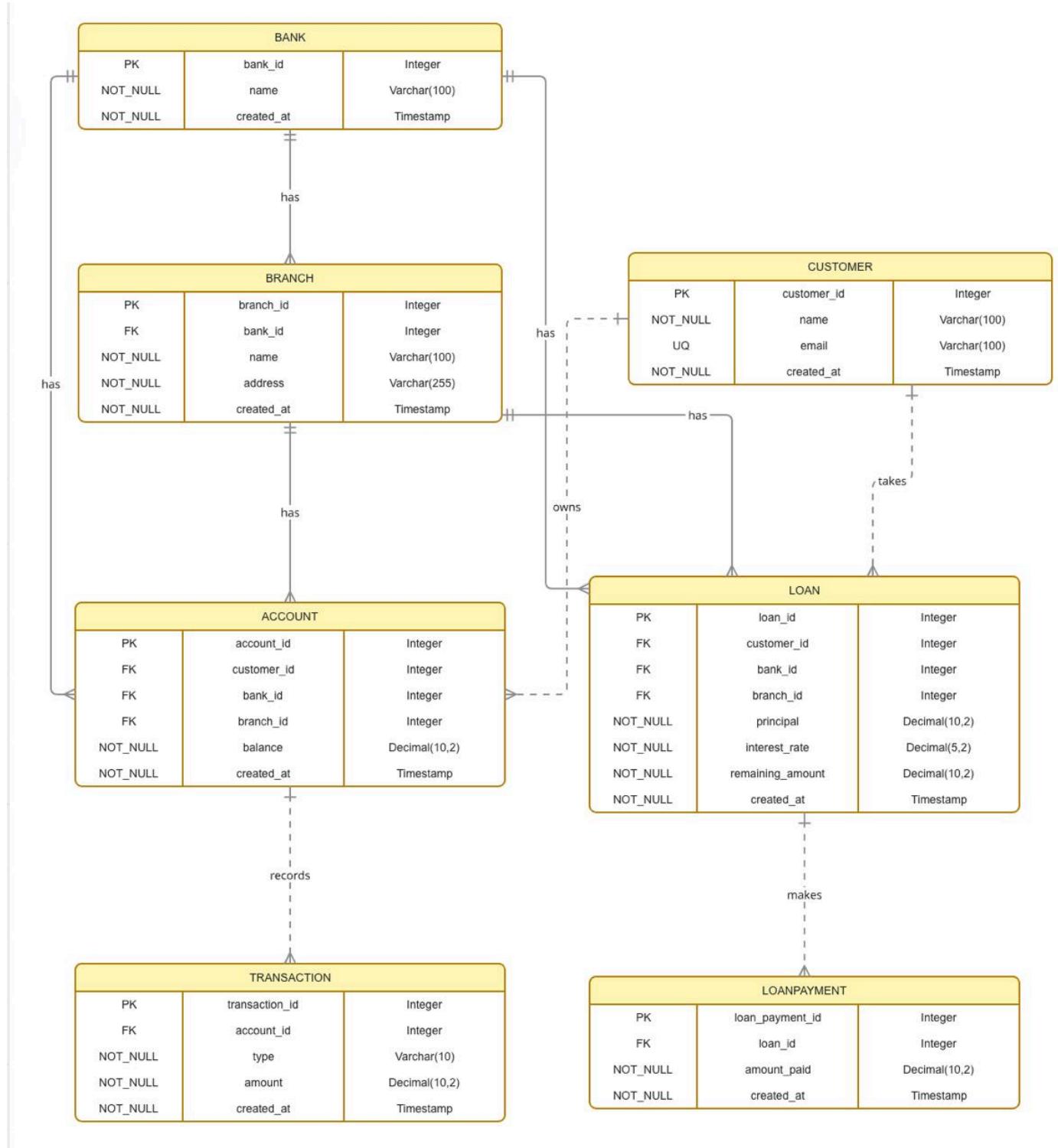
Customer can see his transactions-----> API, Database Connection, Business Logic

Customer can see his loan details -----> API, Database Connection, Business Logic

So for now i see relationships so now I DRAW ER diagram:

Here is what i thought of relationships:

- One Bank has many Branches (1:N)
- One Bank has many Accounts (1:N)
- One Bank has many Loans (1:N)
- One Branch has many Accounts (1:N)
- One Branch has many Loans (1:N)
- One Customer can have many Accounts, and each Account belongs to one Customer (1:N)
- One Customer can have many Loans, and each Loan belongs to one Customer (1:N)
- One Account has many Transactions (1:N)
- One Loan has many LoanPayments (1:N)
- Customer to Bank and Customer to Branch relationships are many-to-many and are achieved indirectly through the Account and Loan entities.



Now after Drawing Er diagram I need to create models that are structs in go and use GORM library so that we can establish GORM->Postgres Connection and tables get autofilled whenever user enters json data after jsonbinding in these structs and from these structs to postgres tables

Now we need 7 GORM models as we have 7 entities 7 postgres tables to fill:

1) Account.go->

The screenshot shows a code editor interface with a dark theme. On the left is an Explorer sidebar showing a project structure under 'BANKING-SYSTEM'. The 'models' folder contains several files: account.go (selected), bank.go, branch.go, customer.go, loan_payment.go, loan.go, and transaction.go. Other parts of the project include cmd/server, internal, config, db, migrate.go, postgres.go, handlers, and services. Configuration files .env, go.mod, and go.sum are also listed. The main editor area displays the code for 'account.go':

```
internal > models > account.go > Account > Branch
1 package models
2
3 import "time"
4
5 type Account struct {
6     AccountID uint `gorm:"column:account_id;primaryKey"`
7     CustomerID uint `gorm:"column:customer_id;not null"`
8     BankID uint `gorm:"column:bank_id;not null"`
9     BranchID uint `gorm:"column:branch_id;not null"`
10    Balance float64 `gorm:"column:balance;not null"`
11    CreatedAt time.Time `gorm:"column:created_at"`
12    Type string `gorm:"column:type;default:'SAVINGS'"`
13    Customer Customer `gorm:"foreignKey:CustomerID"`
14    Branch Branch `gorm:"foreignKey:BranchID"`
15
16    Transactions []Transaction `gorm:"foreignKey:AccountID"`
17 }
18
19 func (Account) TableName() string { return "accounts" }
20
```

2) Bank.go->

The screenshot shows a code editor interface with a dark theme. On the left is an Explorer sidebar showing a project structure under 'BANKING-SYSTEM'. The 'models' folder contains several files: account.go, bank.go (selected), branch.go, customer.go, loan_payment.go, loan.go, and transaction.go. Other parts of the project include cmd/server, internal, config, db, migrate.go, postgres.go, handlers, and services. Configuration files .env, go.mod, and go.sum are also listed. The main editor area displays the code for 'bank.go':

```
internal > models > bank.go > ...
1 package models
2
3 import "time"
4
5 type Bank struct {
6     BankID uint `gorm:"column:bank_id;primaryKey"`
7     Name string `gorm:"column:name;not null"`
8     CreatedAt time.Time `gorm:"column:created_at"`
9
10    Branches []Branch `gorm:"foreignKey:BankID"`
11    Loans []Loan `gorm:"foreignKey:BankID"`
12 }
13
14 func (Bank) TableName() string { return "banks" }
15
```

3) Branch.go->

The screenshot shows the VS Code interface with the title bar "banking-system". The left sidebar is titled "EXPLORER" and shows a tree view of the project structure under "BANKING-SYSTEM". The "models" folder contains several files: account.go, branch.go (which is selected), bank.go, customer.go, loan_payment.go, loan.go, and transaction.go. Other sections like cmd/server, internal, config, db, handlers, models, repositories, services, and .env are also listed. The bottom of the sidebar shows ".env", "all_code_snapshot.txt", "go.mod", and "go.sum". The main editor area shows the code for "branch.go".

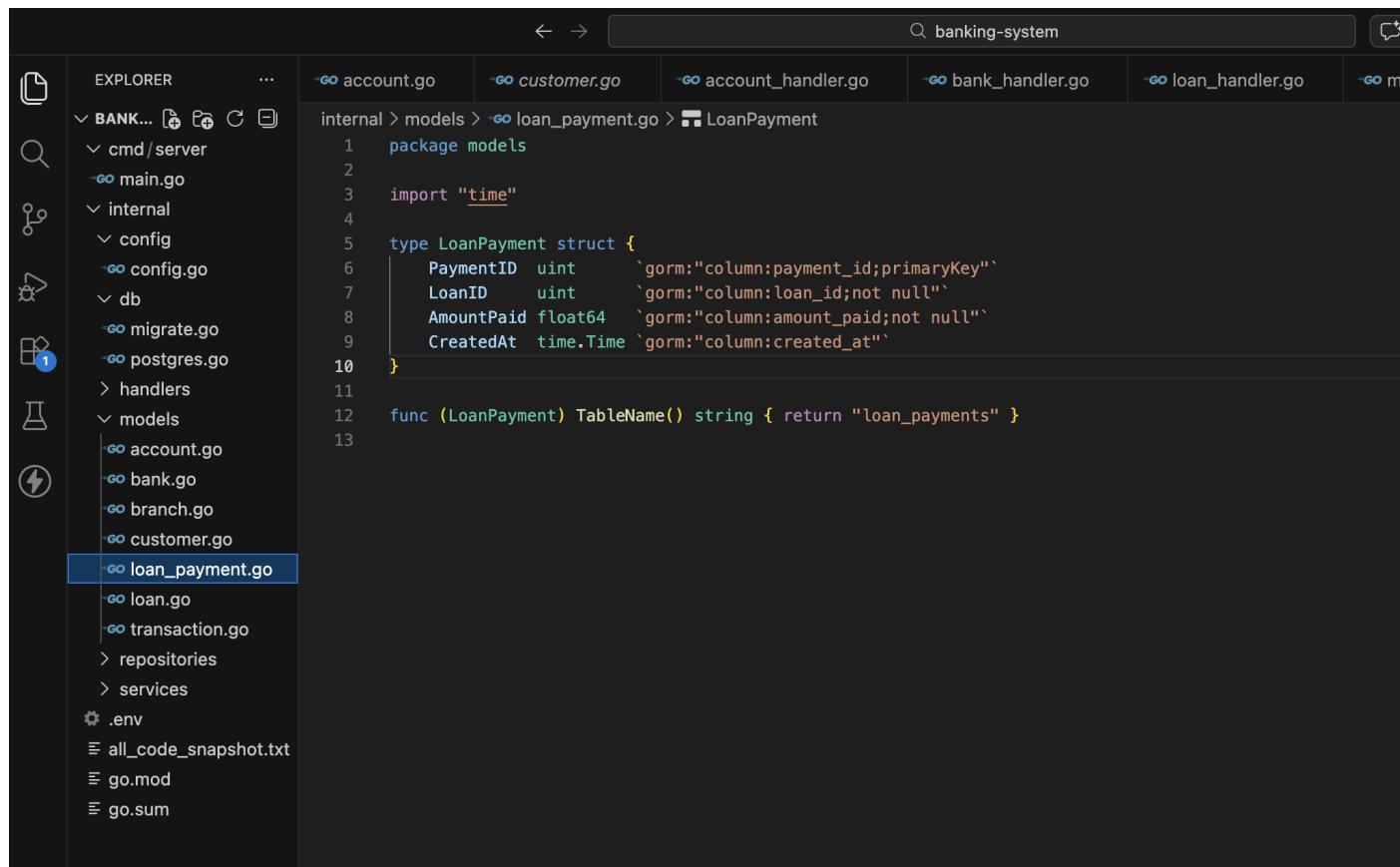
```
internal > models > branch.go > ...
1 package models
2
3 import "time"
4
5 type Branch struct {
6     BranchID uint      `gorm:"column:branch_id;primaryKey"`
7     BankID   uint      `gorm:"column:bank_id;not null"`
8     Name     string    `gorm:"column:name;not null"`
9     Address  string    `gorm:"column:address;not null"`
10    CreatedAt time.Time `gorm:"column:created_at"`
11
12    Bank     Bank      `gorm:"foreignKey:BankID;references:BankID"`
13    Accounts []Account `gorm:"foreignKey:BranchID"`
14    Loans    []Loan    `gorm:"foreignKey:BranchID"`
15 }
16
17 func (Branch) TableName() string { return "branches" }
18 |
```

4) Customer.go->

The screenshot shows the VS Code interface with the title bar "banking-system". The left sidebar is titled "EXPLORER" and shows a tree view of the project structure under "BANKING-SYSTEM". The "models" folder contains several files: account.go, bank.go, branch.go, and customer.go (which is selected). Other files like cmd/server, internal, config, db, handlers, repositories, services, and .env are also listed. The bottom of the sidebar shows ".env", "all_code_snapshot.txt", "go.mod", and "go.sum". The main editor area shows the code for "customer.go".

```
internal > models > customer.go > ...
1 package models
2
3 import "time"
4
5 type Customer struct {
6     CustomerID uint      `gorm:"column:customer_id;primaryKey"`
7     Name       string    `gorm:"column:name;not null"`
8     Email      string    `gorm:"column:email;unique;not null"`
9     CreatedAt time.Time `gorm:"column:created_at"`
10
11    Accounts []Account `gorm:"foreignKey:CustomerID"`
12    Loans    []Loan    `gorm:"foreignKey:CustomerID"`
13 }
14
15 func (Customer) TableName() string { return "customers" }
16 |
```

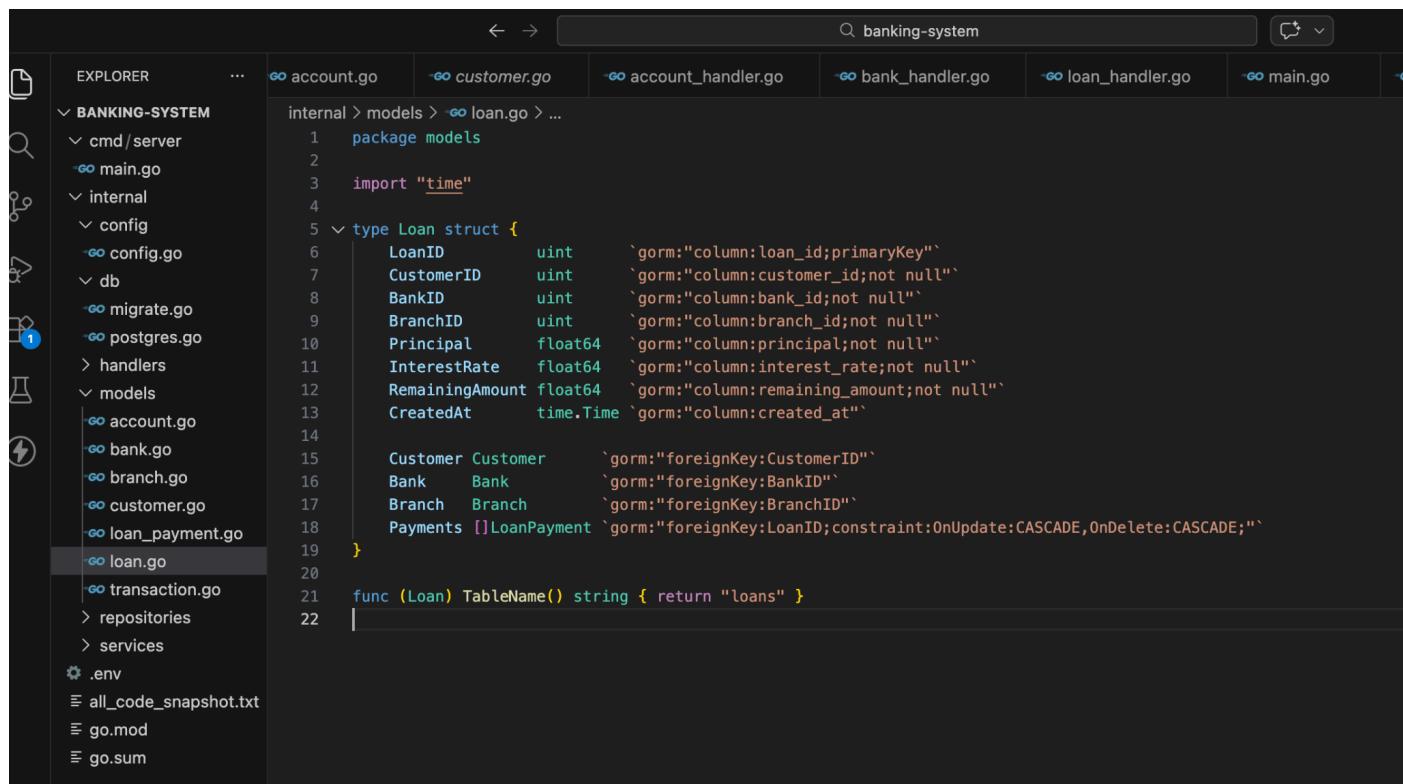
5)loan_payment.go->



```
-> account.go    -> customer.go    -> account_handler.go    -> bank_handler.go    -> loan_handler.go    -> m...
EXPLORER ... -> account.go    -> customer.go    -> account_handler.go    -> bank_handler.go    -> loan_handler.go    -> m...
BANK... cmd/server internal > models > loan_payment.go LoanPayment
  main.go
  internal
    config
      config.go
    db
      migrate.go
    postgres.go
      handlers
        models
          account.go
          bank.go
          branch.go
          customer.go
          loan_payment.go
          loan.go
          transaction.go
      repositories
      services
    .env
  all_code_snapshot.txt
  go.mod
  go.sum
```

```
internal > models > loan_payment.go > LoanPayment
1 package models
2
3 import "time"
4
5 type LoanPayment struct {
6     PaymentID uint      `gorm:"column:payment_id;primaryKey"`
7     LoanID    uint      `gorm:"column:loan_id;not null"`
8     AmountPaid float64   `gorm:"column:amount_paid;not null"`
9     CreatedAt time.Time `gorm:"column:created_at"`
10 }
11
12 func (LoanPayment) TableName() string { return "loan_payments" }
13
```

6)loan.go->



```
-> account.go    -> customer.go    -> account_handler.go    -> bank_handler.go    -> loan_handler.go    -> main.go    -> m...
EXPLORER ... -> account.go    -> customer.go    -> account_handler.go    -> bank_handler.go    -> loan_handler.go    -> main.go    -> m...
BANKING-SYSTEM cmd/server internal > models > loan.go ...
  main.go
  internal
    config
      config.go
    db
      migrate.go
    postgres.go
      handlers
        models
          account.go
          bank.go
          branch.go
          customer.go
          loan_payment.go
          loan.go
          transaction.go
      repositories
      services
    .env
  all_code_snapshot.txt
  go.mod
  go.sum
```

```
internal > models > loan.go ...
1 package models
2
3 import "time"
4
5 type Loan struct {
6     LoanID    uint      `gorm:"column:loan_id;primaryKey"`
7     CustomerID uint      `gorm:"column:customer_id;not null"`
8     BankID    uint      `gorm:"column:bank_id;not null"`
9     BranchID  uint      `gorm:"column:branch_id;not null"`
10    Principal  float64   `gorm:"column:principal;not null"`
11    InterestRate float64   `gorm:"column:interest_rate;not null"`
12    RemainingAmount float64   `gorm:"column:remaining_amount;not null"`
13    CreatedAt time.Time `gorm:"column:created_at"`
14
15    Customer Customer   `gorm:"foreignKey:CustomerID"`
16    Bank     Bank       `gorm:"foreignKey:BankID"`
17    Branch  Branch    `gorm:"foreignKey:BranchID"`
18    Payments []LoanPayment `gorm:"foreignKey:LoanID;constraint:OnUpdate:CASCADE,OnDelete:CASCADE;"`
```

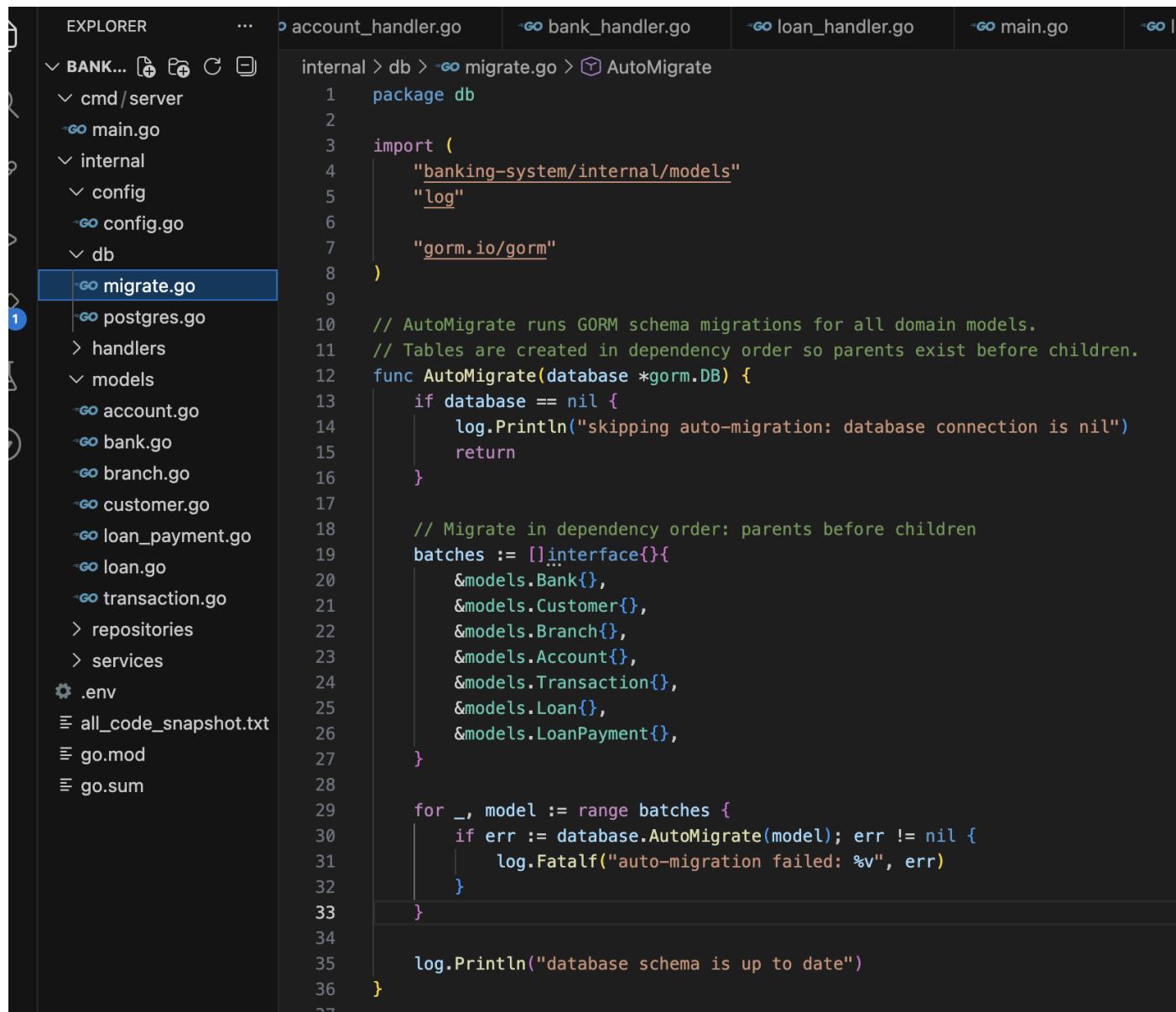
```
func (Loan) TableName() string { return "loans" }
```

7)transaction.go->

The screenshot shows a code editor interface with a sidebar on the left and the main editor area on the right. The sidebar contains icons for file operations like Open, Save, Find, and Refresh, followed by a tree view of the project structure under 'BANKING-SYSTEM'. The main editor area has a header bar with back/forward buttons and a search field containing 'banking-system'. The code editor displays the 'transaction.go' file with syntax highlighting for Go code. The code defines a 'Transaction' struct with fields: TransactionID, AccountID, Type, Amount, and CreatedAt, along with a TableName method.

```
internal > models > -eo transaction.go > ...
1 package models
2
3 import "time"
4
5 type Transaction struct {
6     TransactionID uint `gorm:"column:transaction_id;primaryKey"`
7     AccountID    uint `gorm:"column:account_id;not null"`
8     Type          string `gorm:"column:type;not null"`
9     Amount        float64 `gorm:"column:amount;not null"`
10    CreatedAt    time.Time `gorm:"column:created_at"`
11
12    Account Account `gorm:"foreignKey:AccountID"`
13
14
15 func (Transaction) TableName() string { return "transactions" }
16
```

8)automigrate.go->



```
internal > db > -go migrate.go > AutoMigrate
1 package db
2
3 import (
4     "banking-system/internal/models"
5     "log"
6
7     "gorm.io/gorm"
8 )
9
10 // AutoMigrate runs GORM schema migrations for all domain models.
11 // Tables are created in dependency order so parents exist before children.
12 func AutoMigrate(database *gorm.DB) {
13     if database == nil {
14         log.Println("skipping auto-migration: database connection is nil")
15         return
16     }
17
18     // Migrate in dependency order: parents before children
19     batches := []interface{}{
20         &models.Bank{},
21         &models.Customer{},
22         &models.Branch{},
23         &models.Account{},
24         &models.Transaction{},
25         &models.Loan{},
26         &models.LoanPayment{},
27     }
28
29     for _, model := range batches {
30         if err := database.AutoMigrate(model); err != nil {
31             log.Fatalf("auto-migration failed: %v", err)
32         }
33     }
34
35     log.Println("database schema is up to date")
36 }
37
```

How relationships happen i need foreign keys which i decided like this ->

1.Bank → Branch

A branch cannot exist without bank

So:

branches.bank_id -> banks.bank_id

So branch must belong to a valid bank.

2.Customer → Account

Real-world: A customer can open multiple accounts. An account must belong to exactly one customer.

So:

accounts.customer_id → customers.customer_id

This ensures: No orphan accounts, Clear ownership

3. Account → Bank & Branch

I explicitly linked Account to:

accounts.bank_id → banks.bank_id

accounts.branch_id → branches.branch_id

Because:

A bank can have many branches.

Accounts are opened at a specific branch.

This allows querying: All accounts in a branch All accounts under a bank

4.Transaction → Account

Transactions are dependent records.

Real-world:

- A transaction cannot exist without an account.

So:

transactions.account_id → accounts.account_id

This guarantees:

All transactions are traceable to one account.

5. Loan → Customer + Bank + Branch

Loan belongs to:

- A customer
- A bank
- A branch

So:

loans.customer_id → *customers.customer_id*

loans.bank_id → *banks.bank_id*

loans.branch_id → *branches.branch_id*

This allows:

Querying loans per customer

Loan portfolio per branch

Loan exposure per bank

6. LoanPayment → Loan

Loan payments depend on a loan:

loan_payments.loan_id → *loans.loan_id*

This enforces:

No repayment without a valid loan.

COMPONENTS OF PROJECT:

Handler Layer

What it does

- Handles incoming HTTP requests
- Validates request input (basic format & sanity checks)
- Calls the appropriate method in the **Service layer**
- Converts the result into a structured JSON response (success or error)

It does NOT

- Contain any business logic
- Talk directly to the database

Why we created this separate layer

To achieve clean separation between:

- API / transport concerns
- Business logic

Benefits when things change in future:

- API format changes (v1 → v2)
- Response structure changes
- New authentication middleware added
- New client type (gRPC, CLI, etc.)

→ **Only the handler layer needs to change.**

Service and Repository layers remain untouched.

Example: Deposit Flow

Request

```
text
POST /accounts/1/deposit
{
  "amount": 500
}
```

Inside Handler (pseudo-code)

1. Extract account_id from URL/path
2. Bind & parse JSON body
3. Validate: amount > 0 (simple check)
4. **Call service: h.service.Deposit(accountID, amount)**
5. Return JSON:

Success → { "status": "success", "new_balance": 1500 }

Error → { "error": "insufficient funds", "code": 400 }

Key responsibility (one-liner)

Convert **HTTP input** → Business call

Convert **Business result** → JSON output

Service Layer (Business Logic Layer)

This is the brain of the application.

What it does

- Implements real business rules
- Validates domain-specific rules
- Manages transactions (atomic operations)
- Coordinates calls to one or more repositories

Why we created this layer

Business rules should **never** live in:

- Handlers (HTTP/transport concern)
- Repositories (data access concern)

Examples of business rules that belong here

- Cannot withdraw more than current balance
- Branch must belong to the correct bank
- Loan interest is fixed at 12% per year
- Money transfer must update **both** accounts atomically

Example: Transfer Money

Inside AccountService.Transfer()

1. Validate amount > 0
2. Fetch fromAccount
3. Fetch toAccount
4. Check: fromAccount.balance \geq amount
5. Start database transaction
6. Decrease fromAccount.balance
7. Increase toAccount.balance
8. Create two transaction records (debit + credit)
9. Commit transaction

→ If **anything** fails → rollback everything

→ Guarantees **data consistency** and **atomicity**

Why transactions belong in Service (not Repository)

Because a **transaction = one business operation**

Not a low-level database concern.

Key responsibilities

- Enforce business & domain rules
 - Orchestrate / coordinate repositories
 - Guarantee data consistency
-

Repository Layer (Data Access Layer)

This layer talks directly to the database.

What it does

- Executes GORM (or sqlx / raw SQL) queries
- Performs CRUD operations
- Maps domain models to database tables

Typical methods

- Create(account *Account) error
- GetByID(id uint) (*Account, error)
- UpdateBalance(id uint, newBalance float64) error
- FindByCustomerID(customerID uint) ([]*Account, error)

Why I created this layer

To **isolate** all database-related code.

Future-proof benefit

If tomorrow we need to:

- Switch from PostgreSQL → MySQL
- Replace GORM → raw SQL / sqlx
- Move to MongoDB / DynamoDB

→ **Only repository layer changes**

Service + Handler layers stay almost the same.

API'S IN MY PROJECT:

The screenshot shows a Go code editor interface with the following details:

- File Explorer:** Shows the project structure under "BANKING-SYSTEM".
- Editor:** The main.go file is open, displaying Go code for a banking system.
- Code:** The code initializes services and handlers, defines routes for health, bank, branch, customer, account, and loan services, and runs the application on port 8080.

```
func main() {
    bankService := services.NewBankService(bankRepo)
    branchService := services.NewBranchService(branchRepo)
    customerService := services.NewCustomerService(customerRepo)

    // Handlers
    accountHandler := handlers.NewAccountHandler(accountService)
    loanHandler := handlers.NewLoanHandler(loanService)
    bankHandler := handlers.NewBankHandler(bankService)
    branchHandler := handlers.NewBranchHandler(branchService)
    customerHandler := handlers.NewCustomerHandler(customerService)

    r := gin.Default()

    r.GET("/health", func(c *gin.Context) {
        c.JSON(200, gin.H{"status": "ok"})
    })

    // Bank routes
    r.POST("/banks", bankHandler.CreateBank)
    r.GET("/banks", bankHandler.GetBanks)

    // Branch routes
    r.POST("/branches", branchHandler.CreateBranch)
    r.GET("/branches", branchHandler.GetBranches)

    // Customer routes
    r.POST("/customers", customerHandler.CreateCustomer)
    r.GET("/customers/:id", customerHandler.GetCustomer)

    // Account routes
    r.POST("/accounts", accountHandler.OpenAccount)
    r.POST("/accounts/:id/deposit", accountHandler.Deposit)
    r.POST("/accounts/:id/withdraw", accountHandler.Withdraw)
    r.GET("/accounts/:id", accountHandler.GetAccount)
    r.GET("/accounts/:id/transactions", accountHandler.GetTransactions)
    r.POST("/accounts/transfer", accountHandler.Transfer)

    // Loan routes
    r.POST("/loans", loanHandler.CreateLoan)
    r.POST("/loans/:id/repay", loanHandler.RepayLoan)
    r.GET("/loans/:id", loanHandler.GetLoanDetails)

    r.Run(":8080")
}
```

- Status Bar:** Shows line 70, column 1, tab size 4, UTF-8 encoding, LF line endings, and other status information.

OUTPUT (IMPLEMENTATION OF ALL API'S)

1) CREATING BANK

The screenshot shows the Postman application interface. A POST request is being made to `http://localhost:8080/banks`. The request body contains the following JSON:

```
1 {
2   "name": "State Bank of India"
3 }
```

The response status is `201 Created`, and the response body is:

```
1 {
2   "success": true,
3   "data": {
4     "bank": {
5       "BankID": 1,
6       "Name": "State Bank of India",
7       "CreatedAt": "2026-02-12T10:30:27.118481+05:30",
8       "Branches": null,
9       "Loans": null
10      }
11    }
12 }
```

2) CREATING BRANCH

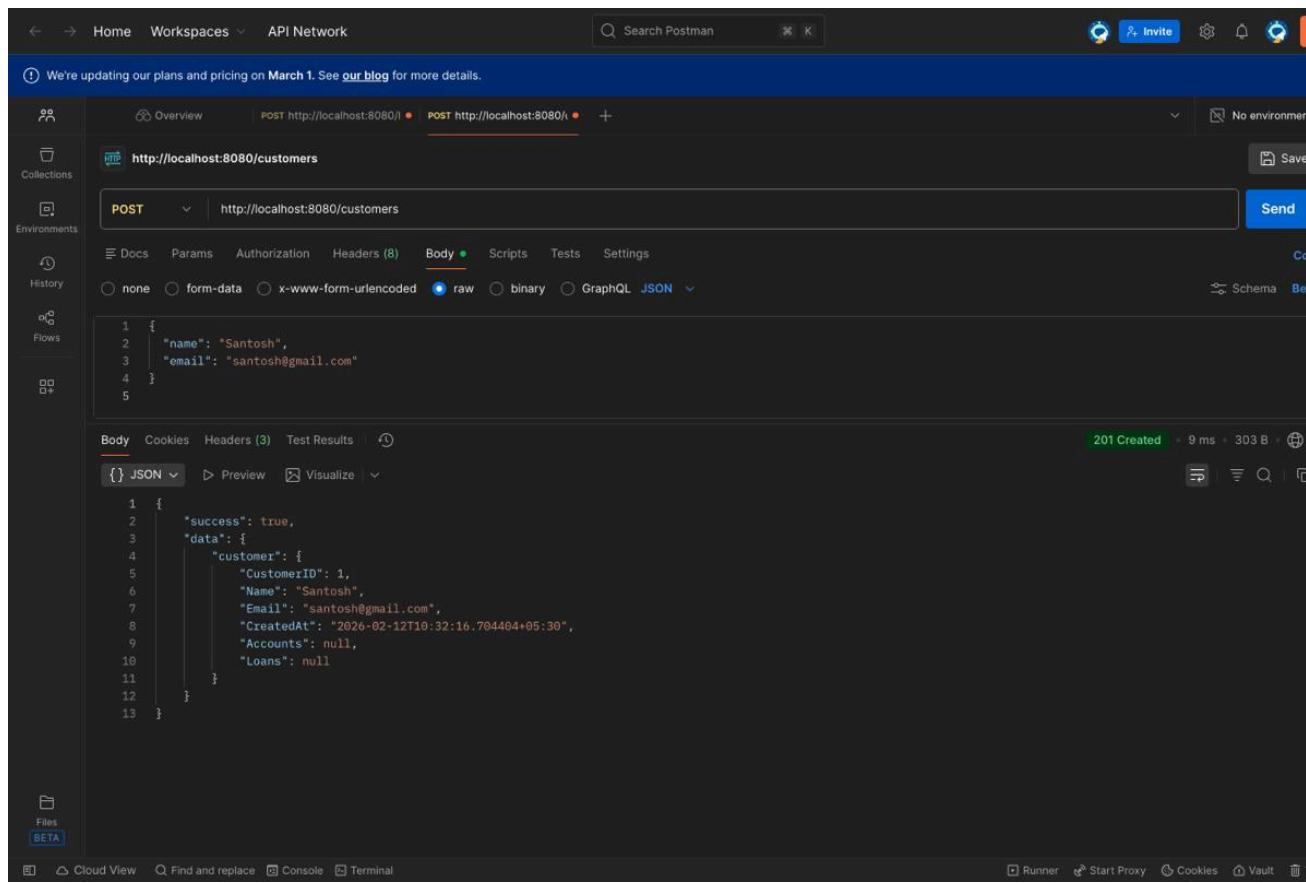
The screenshot shows the Postman application interface. A POST request is being made to `http://localhost:8080/branches`. The request body contains the following JSON:

```
1 {
2   "bank_id": 1,
3   "name": "Mumbai Branch",
4   "address": "Mumbai"
5 }
```

The response status is `201 Created`, and the response body is:

```
1 {
2   "success": true,
3   "data": {
4     "branch": {
5       "BranchID": 1,
6       "BankID": 1,
7       "Name": "Mumbai Branch",
8       "Address": "Mumbai",
9       "CreatedAt": "2026-02-12T10:31:28.800743+05:30",
10      "Bank": {
11        "BankID": 0,
12        "Name": "",
13        "CreatedAt": "0001-01-01T00:00:00Z",
14        "Branches": null,
15        "Loans": null
16      },
17      "Accounts": null,
18      "Loans": null
19    }
20  }
21 }
```

3) CREATING CUSTOMER



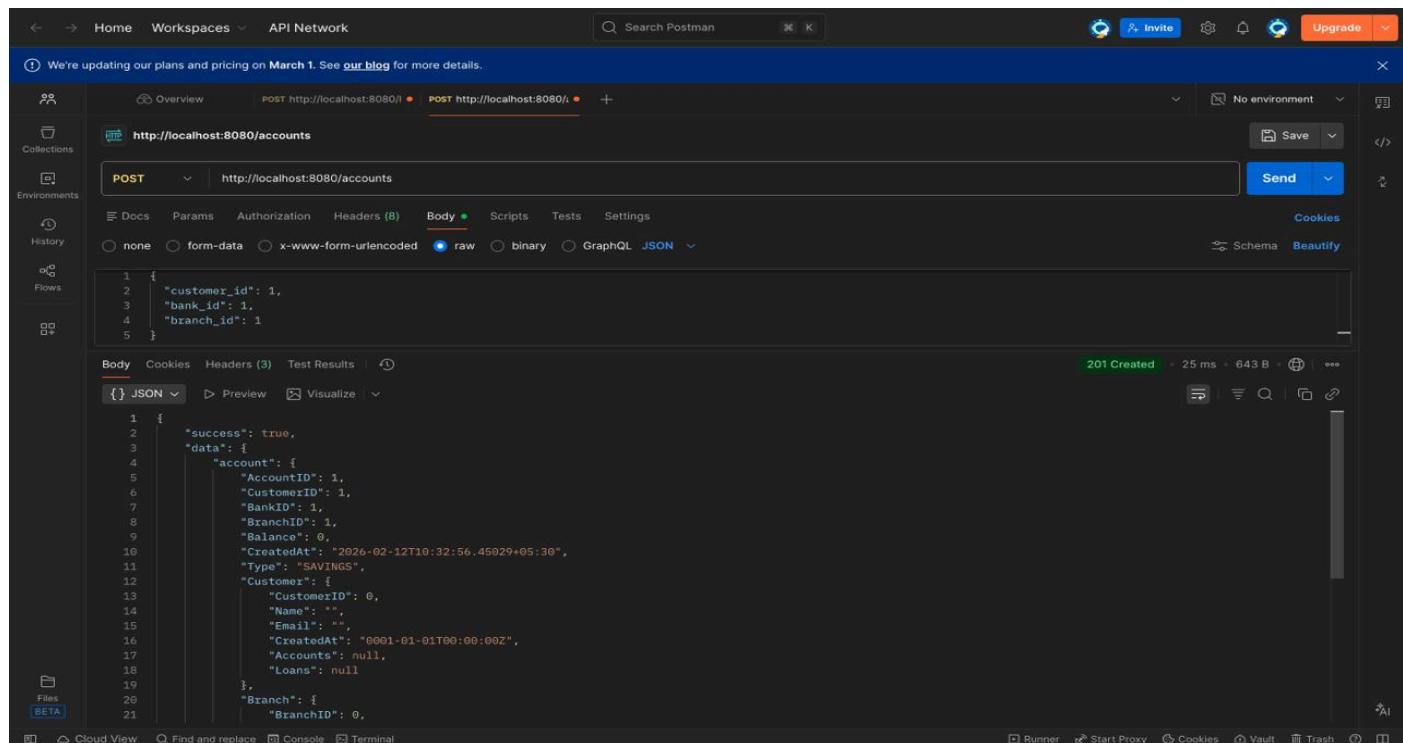
The screenshot shows the Postman application interface. On the left, there's a sidebar with sections for Collections, Environments, History, Flows, and Files (BETA). The main area has tabs for Overview, POST http://localhost:8080/l, and POST http://localhost:8080/n. The current tab is POST http://localhost:8080/customers. The request URL is http://localhost:8080/customers. The method is POST. The Body tab is selected, showing raw JSON input:

```
1 {
2   "name": "Santosh",
3   "email": "santosh@gmail.com"
4 }
```

The response tab shows a 201 Created status with a JSON response:

```
1 {
2   "success": true,
3   "data": {
4     "customer": {
5       "CustomerID": 1,
6       "Name": "Santosh",
7       "Email": "santosh@gmail.com",
8       "Createdat": "2026-02-12T10:32:16.704404+05:30",
9       "Accounts": null,
10      "Loans": null
11    }
12  }
13 }
```

4) OPEN ACCOUNT



The screenshot shows the Postman application interface. On the left, there's a sidebar with sections for Collections, Environments, History, Flows, and Files (BETA). The main area has tabs for Overview, POST http://localhost:8080/l, and POST http://localhost:8080/n. The current tab is POST http://localhost:8080/accounts. The request URL is http://localhost:8080/accounts. The method is POST. The Body tab is selected, showing raw JSON input:

```
1 {
2   "customer_id": 1,
3   "bank_id": 1,
4   "branch_id": 1
5 }
```

The response tab shows a 201 Created status with a JSON response:

```
1 {
2   "success": true,
3   "data": {
4     "account": {
5       "AccountID": 1,
6       "CustomerID": 1,
7       "BankID": 1,
8       "BranchID": 1,
9       "Balance": 0,
10      "Createdat": "2026-02-12T10:32:56.45629+05:30",
11      "Type": "SAVINGS",
12      "Customer": {
13        "CustomerID": 0,
14        "Name": "",
15        "Email": "",
16        "Createdat": "2001-01-01T00:00:00Z",
17        "Accounts": null,
18        "Loans": null
19      },
20      "Branch": {
21        "BranchID": 0,
22      }
23    }
24  }
25 }
```

The screenshot shows the Postman application interface. In the top navigation bar, there are links for Home, Workspaces, API Network, and a search bar labeled "Search Postman". On the right side of the header, there are buttons for Invite, Upgrade, and other account-related options.

The main workspace displays a collection named "Collections" containing a single item "http://localhost:8080/accounts". A POST request is being made to this endpoint. The "Body" tab is selected, showing the raw JSON payload:

```
1 {
2   "customer_id": 1,
3   "bank_id": 1,
4   "branch_id": 1
5 }
```

Below the body, the "Test Results" section shows a successful response with a status of 201 Created, a duration of 25 ms, and a response size of 643 B. The response body is partially visible, showing nested objects for Branch, Bank, and Transactions.

6) DEPOSIT MONEY -> SANTOSH DEPOSIT 5000

This screenshot shows another POST request in Postman. The URL is "http://localhost:8080/accounts/1/deposit". The "Body" tab is selected, showing the raw JSON payload:

```
1 {
2   "amount": 5000
3 }
```

The response status is 200 OK, with a duration of 48 ms and a response size of 710 B. The response body is a detailed JSON object containing information about the deposit transaction, including the account ID, customer ID, bank ID, branch ID, balance, creation date, type (SAVINGS), and customer details like name and email.

The screenshot shows the Postman interface with a collection named 'Collections'. A POST request is made to `http://localhost:8080/accounts/1/deposit`. The request body is set to raw JSON with the value `{"amount": 5000}`. The response status is 200 OK, and the response body is displayed as:

```
22     "BankID": 0,
23     "Name": "",
24     "Address": "",
25     "Createdat": "0001-01-01T00:00:00Z",
26     "Bank": {
27       "BankID": 0,
28       "Name": "",
29       "Createdat": "0001-01-01T00:00:00Z",
30       "Branches": null,
31       "Loans": null
32     },
33     "Accounts": null,
34     "Loans": null
35   },
36   "Transactions": null
37 },
38   "amount": 5000,
39   "message": "deposit successful",
40   "transaction": "DEPOSIT"
41 }
```

7) Withdraw Money -> SANTOSH WITHDRAWS 1000, NOW HAS 4000

The screenshot shows the Postman interface with a collection named 'Collections'. A POST request is made to `http://localhost:8080/accounts/1/withdraw`. The request body is set to raw JSON with the value `{"amount": 1000}`. The response status is 200 OK, and the response body is displayed as:

```
1   {
2     "success": true,
3     "data": {
4       "account": {
5         "AccountID": 1,
6         "CustomerID": 1,
7         "BankID": 1,
8         "BranchID": 1,
9         "Balance": 4000,
10        "Createdat": "2026-02-12T10:32:56.45029+05:30",
11        "Type": "SAVINGS",
12        "Customer": {
13          "CustomerID": 0,
14          "Name": "",
15          "Email": "",
16          "Createdat": "0001-01-01T00:00:00Z",
17          "Accounts": null,
18          "Loans": null
19        },
20        "Branch": {
21          "BranchID": 0,
22        }
23      }
24    }
25 }
```

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:8080/accounts/1/withdraw`. The request method is `POST`. The body contains the following JSON:

```
1 {  
2   "amount": 1000  
3 }  
4
```

The response status is `200 OK` with a response time of 9 ms and a size of 712 B. The response body is:

```
20 {  
21   "Branch": {  
22     "BranchID": 0,  
23     "BankID": 0,  
24     "Name": "",  
25     "Address": "",  
26     "CreatedAt": "0001-01-01T00:00:00Z",  
27     "Bank": {  
28       "BankID": 0,  
29       "Name": "",  
30       "CreatedAt": "0001-01-01T00:00:00Z",  
31       "Branches": null,  
32       "Loans": null  
33     },  
34     "Accounts": null,  
35     "Loans": null  
36   },  
37   "Transactions": null  
38 },  
39   "amount": 1000,  
40   "message": "withdraw successful",  
41   "transaction": "WITHDRAW"  
42 }
```

8) NOW TO SEE ALL TRANSACTIONS MADE

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:8080/accounts/1/transactions`. The request method is `GET`. The response status is `200 OK` with a response time of 20 ms and a size of 1.29 KB. The response body is:

```
1 {  
2   "success": true,  
3   "data": [  
4     {  
5       "TransactionID": 2,  
6       "AccountId": 1,  
7       "Type": "WITHDRAW",  
8       "Amount": 1000,  
9       "CreatedAt": "2026-02-12T10:36:16.450521+05:30",  
10      "Account": {  
11        "AccountID": 0,  
12        "CustomerID": 0,  
13        "BankID": 0,  
14        "BranchID": 0,  
15        "Balance": 0,  
16        "CreatedAt": "0001-01-01T00:00:00Z",  
17        "Type": "",  
18        "Customer": {  
19          "CustomerID": 0,  
20          "Name": "",  
21          "Email": "",  
22          "CreatedAt": "0001-01-01T00:00:00Z",  
23          "Accounts": null,  
24          "Loans": null  
25        },  
26        "Branch": {  
27          "BranchID": 0,  
28          "BankID": 0,  
29          "Name": ""  
30        }  
31      }  
32    }  
33  ]  
34 }
```

Home Workspaces API Network

① We're updating our plans and pricing on March 1. See our blog for more details.

http://localhost:8080/accounts/1/transactions

GET http://localhost:8080/accounts/1/transactions

Body Cookies Headers (3) Test Results

{} JSON Preview Visualize

```
26     "Branch": {  
27         "BranchID": 0,  
28         "BankID": 0,  
29         "Name": "",  
30         "Address": "",  
31         "CreatedAt": "0001-01-01T00:00:00Z",  
32         "Bank": {  
33             "BankID": 0,  
34             "Name": "",  
35             "CreatedAt": "0001-01-01T00:00:00Z",  
36             "Branches": null,  
37             "Loans": null  
38         },  
39         "Accounts": null,  
40         "Loans": null  
41     },  
42     "Transactions": null  
43 },  
44 {  
45     "TransactionID": 1,  
46     "AccountID": 1,  
47     "Type": "DEPOSIT",  
48     "Amount": 5000,  
49     "Createdat": "2026-02-12T10:34:54.435239+05:30",  
50     "Account": {  
51         "AccountID": 0,  
52         "CustomerID": 0,  
53         "BankID": 0,  
54         "BranchID": 0  
55     },  
56     "Branch": {  
57         "BranchID": 0,  
58         "BankID": 0,  
59         "Name": "",  
60         "Address": "",  
61         "CreatedAt": "0001-01-01T00:00:00Z",  
62         "Bank": {  
63             "BankID": 0,  
64             "Name": "",  
65             "Address": "",  
66             "CreatedAt": "0001-01-01T00:00:00Z",  
67             "Branch": {  
68                 "BranchID": 0,  
69                 "BankID": 0,  
70                 "Name": "",  
71                 "Address": "",  
72                 "CreatedAt": "0001-01-01T00:00:00Z",  
73                 "Bank": {  
74                     "BankID": 0,  
75                     "Name": "",  
76                     "Address": "",  
77                     "CreatedAt": "0001-01-01T00:00:00Z",  
78                     "Branch": {  
79                         "BranchID": 0,  
80                         "BankID": 0,  
81                         "Name": "",  
82                         "Address": "",  
83                         "CreatedAt": "0001-01-01T00:00:00Z",  
84                         "Bank": {  
85                             "BankID": 0,  
86                             "Name": "",  
87                             "Address": "",  
88                             "CreatedAt": "0001-01-01T00:00:00Z",  
89                             "Branch": {  
90                                 "BranchID": 0,  
91                                 "BankID": 0,  
92                                 "Name": "",  
93                                 "Address": "",  
94                                 "CreatedAt": "0001-01-01T00:00:00Z",  
95                                 "Bank": {  
96                                     "BankID": 0,  
97                                     "Name": "",  
98                                     "Address": "",  
99                                     "CreatedAt": "0001-01-01T00:00:00Z"  
100                                 }  
101                             }  
102                         }  
103                     }  
104                 }  
105             }  
106         }  
107     }  
108 }
```

200 OK 20 ms 1.29 KB

Home Workspaces API Network

① We're updating our plans and pricing on March 1. See our blog for more details.

http://localhost:8080/accounts/1/transactions

GET http://localhost:8080/accounts/1/transactions

Body Cookies Headers (3) Test Results

{} JSON Preview Visualize

```
45     "TransactionID": 1,  
46     "AccountID": 1,  
47     "Type": "DEPOSIT",  
48     "Amount": 5000,  
49     "Createdat": "2026-02-12T10:34:54.435239+05:30",  
50     "Account": {  
51         "AccountID": 0,  
52         "CustomerID": 0,  
53         "BankID": 0,  
54         "BranchID": 0,  
55         "Balance": 0,  
56         "Createdat": "0001-01-01T00:00:00Z",  
57         "Type": "",  
58         "Customer": {  
59             "CustomerID": 0,  
60             "Name": "",  
61             "Email": "",  
62             "Createdat": "0001-01-01T00:00:00Z",  
63             "Accounts": null,  
64             "Loans": null  
65         },  
66         "Branch": {  
67             "BranchID": 0,  
68             "BankID": 0,  
69             "Name": "",  
70             "Address": "",  
71             "CreatedAt": "0001-01-01T00:00:00Z",  
72             "Bank": {  
73                 "BankID": 0,  
74                 "Name": "",  
75                 "Address": "",  
76                 "CreatedAt": "0001-01-01T00:00:00Z",  
77                 "Branch": {  
78                     "BranchID": 0,  
79                     "BankID": 0,  
80                     "Name": "",  
81                     "Address": "",  
82                     "CreatedAt": "0001-01-01T00:00:00Z",  
83                     "Bank": {  
84                         "BankID": 0,  
85                         "Name": "",  
86                         "Address": "",  
87                         "CreatedAt": "0001-01-01T00:00:00Z",  
88                         "Branch": {  
89                             "BranchID": 0,  
90                             "BankID": 0,  
91                             "Name": "",  
92                             "Address": "",  
93                             "CreatedAt": "0001-01-01T00:00:00Z",  
94                             "Bank": {  
95                                 "BankID": 0,  
96                                 "Name": "",  
97                                 "Address": "",  
98                                 "CreatedAt": "0001-01-01T00:00:00Z",  
99                                 "Branch": {  
100                                     "BranchID": 0,  
101                                     "BankID": 0,  
102                                     "Name": "",  
103                                     "Address": "",  
104                                     "CreatedAt": "0001-01-01T00:00:00Z"  
105                                 }  
106                             }  
107                         }  
108                     }  
109                 }  
110             }  
111         }  
112     }  
113 }
```

200 OK 20 ms 1.29 KB

The screenshot shows the Postman application interface. In the center, there is a request card for a GET request to `http://localhost:8080/accounts/1/transactions`. The response status is 200 OK with a 20 ms duration and 1.29 KB size. The response body is displayed as JSON:

```
59     "Customer": {
60       "CustomerID": 0,
61       "Name": "",
62       "Email": "",
63       "CreatedAt": "0001-01-01T00:00:00Z",
64       "Accounts": null,
65       "Loans": null
66     },
67     "Branch": {
68       "BranchID": 0,
69       "BankID": 0,
70       "Name": "",
71       "Address": "",
72       "CreatedAt": "0001-01-01T00:00:00Z",
73       "Bank": {
74         "BankID": 0,
75         "Name": "",
76         "CreatedAt": "0001-01-01T00:00:00Z",
77         "Branches": null,
78         "Loans": null
79       },
80       "Accounts": null,
81       "Loans": null
82     },
83     "Transactions": null
84   }
85 }
```

**NOW TO TRANSFER BETWEEN 2 ACCOUNTS
WE ALREADY HAVE SANTOSH CUSTOMER , NOW CREATING AVINASH CUSTOMER**

A) CREATING AVINASH CUSTOMER

The screenshot shows the Postman application interface. In the center, there is a request card for a POST request to `http://localhost:8080/customers`. The response status is 201 Created with a 98 ms duration and 303 B size. The response body is displayed as JSON:

```
1   {
2     "name": "Avinash",
3     "email": "avinash@gmail.com"
4   }
5 
```

Body

```
1   {
2     "success": true,
3     "data": {
4       "customer": {
5         "CustomerID": 2,
6         "Name": "Avinash",
7         "Email": "avinash@gmail.com",
8         "CreatedAt": "2026-02-12T16:07:09.592958+05:30",
9         "Accounts": null,
10        "Loans": null
11      }
12    }
13 }
```

B) Opening Avinash Account in Same Bank And Branch

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:8080/accounts`. The request method is `POST`. The response status is `201 Created` with a response time of `130 ms` and a size of `644 B`. The response body is a JSON object:

```
1 {  
2   "success": true,  
3   "data": {  
4     "account": {  
5       "AccountID": 2,  
6       "CustomerID": 2,  
7       "BankID": 1,  
8       "BranchID": 1,  
9       "Balance": 0,  
10      "CreatedAt": "2026-02-12T16:08:34.875083+05:30",  
11      "Type": "SAVINGS",  
12      "Customer": {  
13        "CustomerID": 0,  
14        "Name": "",  
15        "Email": "",  
16        "CreatedAt": "0001-01-01T00:00:00Z",  
17        "Accounts": null,  
18        "Loans": null  
19      }  
20    },  
21    "Branch": {  
22      "BranchID": 0,  
23      "BankID": 0,  
24      "Name": "",  
25      "Address": "",  
26      "CreatedAt": "0001-01-01T00:00:00Z",  
27      "Bank": {  
28        "BankID": 0,  
29        "Name": "",  
30        "CreatedAt": "0001-01-01T00:00:00Z",  
31        "Branches": null,  
32        "Loans": null  
33      },  
34      "Accounts": null,  
35      "Loans": null  
36    },  
37    "Transactions": null  
38  }  
}
```

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:8080/accounts`. The request method is `POST`. The response status is `201 Created` with a response time of `130 ms` and a size of `644 B`. The response body is a JSON object:

```
1 {  
2   "success": true,  
3   "data": {  
4     "account": {  
5       "AccountID": 2,  
6       "CustomerID": 2,  
7       "BankID": 1,  
8       "BranchID": 1,  
9       "Balance": 0,  
10      "CreatedAt": "2026-02-12T16:08:34.875083+05:30",  
11      "Type": "SAVINGS",  
12      "Customer": {  
13        "CustomerID": 0,  
14        "Name": "",  
15        "Email": "",  
16        "CreatedAt": "0001-01-01T00:00:00Z",  
17        "Accounts": null,  
18        "Loans": null  
19      }  
20    },  
21    "Branch": {  
22      "BranchID": 0,  
23      "BankID": 0,  
24      "Name": "",  
25      "Address": "",  
26      "CreatedAt": "0001-01-01T00:00:00Z",  
27      "Bank": {  
28        "BankID": 0,  
29        "Name": "",  
30        "CreatedAt": "0001-01-01T00:00:00Z",  
31        "Branches": null,  
32        "Loans": null  
33      },  
34      "Accounts": null,  
35      "Loans": null  
36    },  
37    "Transactions": null  
38  }  
}
```

C) NOW SANTOSH TRANSFERS 1000 FROM HIS ACCOUNT TO AVINASH

```
{  
    "success": true,  
    "data": {  
        "amount": 1000,  
        "from_account": {  
            "AccountID": 1,  
            "CustomerID": 1,  
            "BankID": 1,  
            "BranchID": 1,  
            "Balance": 3000,  
            "CreatedAt": "2026-02-12T10:32:56.45029+05:30",  
            "Type": "SAVINGS",  
            "Customer": {  
                "CustomerID": 0,  
                "Name": "",  
                "Email": "",  
                "CreatedAt": "0001-01-01T00:00:00Z",  
                "Accounts": null,  
                "Loans": null  
            },  
            "Branch": {  
                "BranchID": 0,  
                "BankID": 0,  
                "Name": "",  
                "Address": "",  
                "CreatedAt": "0001-01-01T00:00:00Z",  
                "Bank": {  
                    "BankID": 0,  
                    "Name": "",  
                    "CreatedAt": "0001-01-01T00:00:00Z",  
                    "Branches": null,  
                    "Loans": null  
                },  
                "Accounts": null,  
                "Loans": null  
            },  
            "Transactions": null  
        },  
        "message": "transfer successful",  
        "to_account": {  
            "AccountID": 2,  
            "CustomerID": 2,  
            "BankID": 1,  
            "BranchID": 1,  
            "Balance": 1000,  
            "Type": "SAVINGS",  
            "Customer": {  
                "CustomerID": 1,  
                "Name": "Avinash",  
                "Email": "avinash@example.com",  
                "CreatedAt": "2026-02-12T10:32:56.45029+05:30",  
                "Accounts": null,  
                "Loans": null  
            },  
            "Branch": {  
                "BranchID": 1,  
                "BankID": 1,  
                "Name": "State Bank of India",  
                "Address": "123 Main Street, Mumbai",  
                "CreatedAt": "2026-02-12T10:32:56.45029+05:30",  
                "Bank": {  
                    "BankID": 1,  
                    "Name": "State Bank of India",  
                    "CreatedAt": "2026-02-12T10:32:56.45029+05:30",  
                    "Branches": null,  
                    "Loans": null  
                },  
                "Accounts": null,  
                "Loans": null  
            },  
            "Transactions": null  
        }  
    }  
}
```

```
"BranchID": 1,
"Balance": 1000,
"CreatedAt": "2026-02-12T16:08:34.875083+05:30",
>Type": "SAVINGS",
"Customer": {
    "CustomerID": 0,
    "Name": "",
    "Email": "",
    "CreatedAt": "0001-01-01T00:00:00Z",
    "Accounts": null,
    "Loans": null
},
"Branch": {
    "BranchID": 0,
    "BankID": 0,
    "Name": "",
    "Address": "",
    "CreatedAt": "0001-01-01T00:00:00Z",
    "Bank": {
        "BankID": 0,
        "Name": "",
        "CreatedAt": "0001-01-01T00:00:00Z",
        "Branches": null,
        "Loans": null
    },
    "Accounts": null,
    "Loans": null
},
"Transactions": null
}
}
}
```

NOW SANTOSH HAS 3000, AVINASH HAS 1000

The screenshot shows the Postman application interface. On the left, there's a sidebar with icons for Collections, Environments, History, Flows, and Files (BETA). The main area has tabs for Overview, POST http://localhost:8080/1, POST http://localhost:8080/2, GET http://localhost:8080/accounts/1 (which is selected), POST http://localhost:8080/3, and GET Untitled Request. A search bar at the top right says "Search Postman". Below the tabs, there's a "Send" button. The main content area shows a "GET" request to "http://localhost:8080/accounts/1". Under "Body", the response is displayed as JSON:

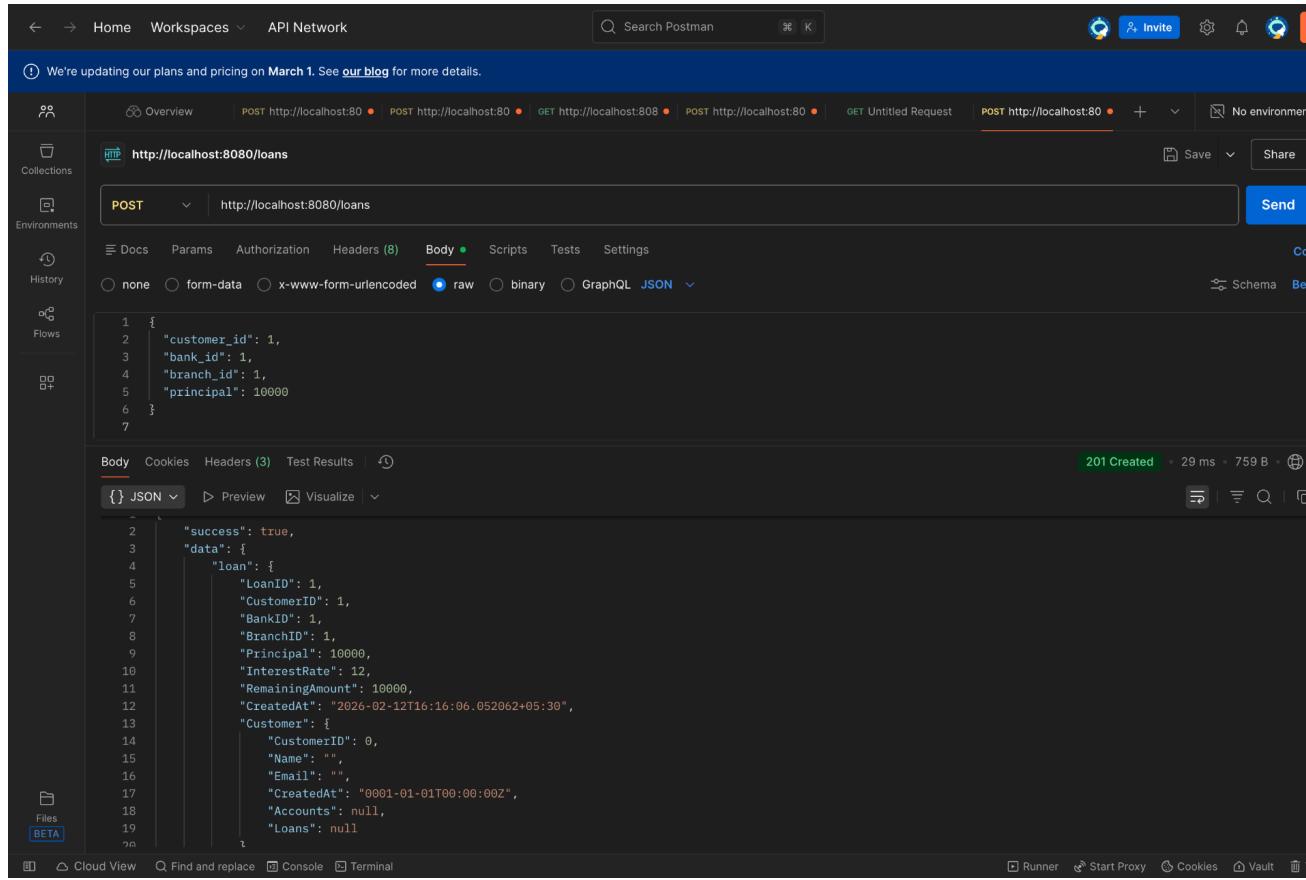
```
1 {
2     "success": true,
3     "data": {
4         "AccountID": 1,
5         "CustomerID": 1,
6         "BankID": 1,
7         "BranchID": 1,
8         "Balance": 3000,
9         "CreatedAt": "2026-02-12T10:32:56.45029+05:30",
10        "Type": "SAVINGS",
11        "Customer": {
12            "CustomerID": 0,
13            "Name": "",
14            "Email": "",
15            "CreatedAt": "0001-01-01T00:00:00Z",
16            "Accounts": null,
17            "Loans": null
18        },
19        "Branch": {
20            "BranchID": 0,
21            "BankID": 0,
22            "Name": "",
23            "Address": "",
24            "CreatedAt": "0001-01-01T00:00:00Z",
25            "Bank": {
26                "BankID": 0,
27                "Name": "",
28                "CreatedAt": "0001-01-01T00:00:00Z",
29                "Branches": null
            }
        }
    }
}
```

This screenshot is nearly identical to the one above, showing the same Postman interface and a successful GET request to "http://localhost:8080/accounts/2". The JSON response is very similar, with only minor differences in the account ID and balance values:

```
1 {
2     "success": true,
3     "data": {
4         "AccountID": 2,
5         "CustomerID": 2,
6         "BankID": 1,
7         "BranchID": 1,
8         "Balance": 1000,
9         "CreatedAt": "2026-02-12T16:08:34.875083+05:30",
10        "Type": "SAVINGS",
11        "Customer": {
12            "CustomerID": 0,
13            "Name": "",
14            "Email": "",
15            "CreatedAt": "0001-01-01T00:00:00Z",
16            "Accounts": null,
17            "Loans": null
18        },
19        "Branch": {
20            "BranchID": 0,
21            "BankID": 0,
22            "Name": "",
23            "Address": "",
24            "CreatedAt": "0001-01-01T00:00:00Z",
25            "Bank": {
26                "BankID": 0,
27                "Name": "",
28                "CreatedAt": "0001-01-01T00:00:00Z",
29                "Branches": null
            }
        }
    }
}
```

NOW TO CREATE LOAN/REPAY LOAN/ GET LOAN DETAILS

A) SANTOSH TAKES A LOAN OF 10,000 (AUTO 12% RATE APPLIED)



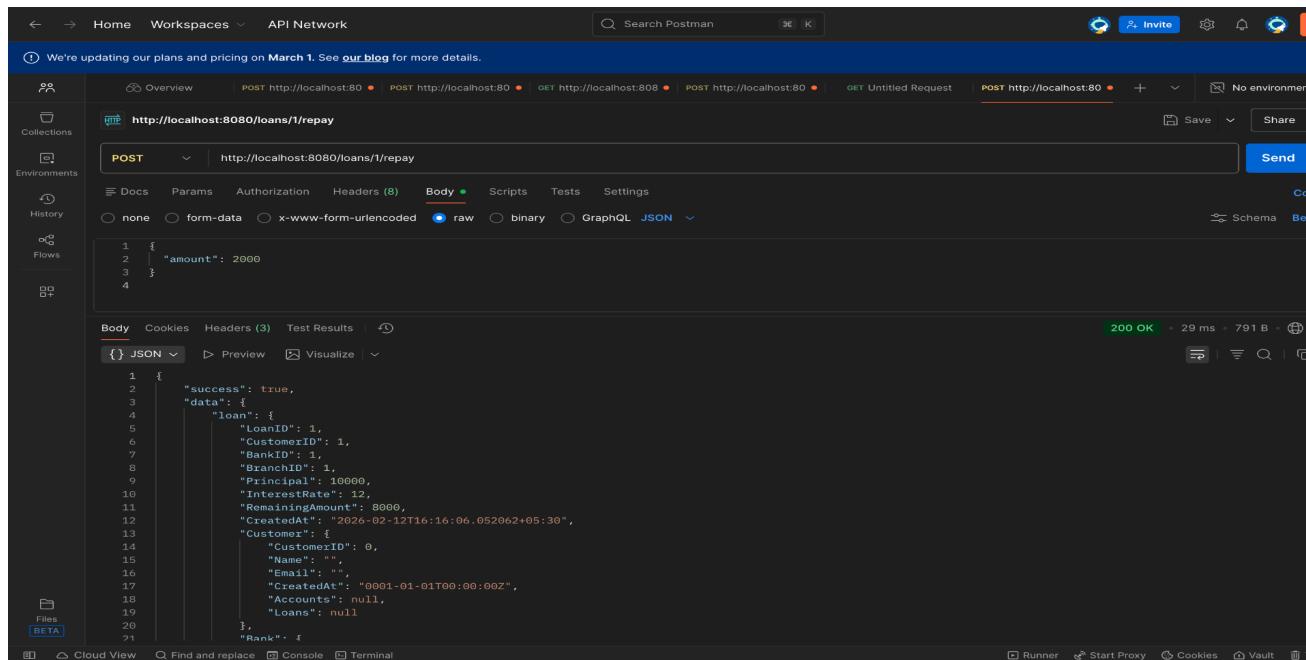
The screenshot shows the Postman interface with a POST request to `http://localhost:8080/loans`. The request body is raw JSON:

```
1 {
2   "customer_id": 1,
3   "bank_id": 1,
4   "branch_id": 1,
5   "principal": 10000
6 }
7
```

The response status is `201 Created` with a response time of `29 ms` and a size of `759 B`. The response body is:

```
2   "success": true,
3   "data": {
4     "loan": {
5       "LoanID": 1,
6       "CustomerID": 1,
7       "BankID": 1,
8       "BranchID": 1,
9       "Principal": 10000,
10      "InterestRate": 12,
11      "RemainingAmount": 10000,
12      "Createdat": "2026-02-12T16:16:06.052062+05:30",
13      "Customer": {
14        "CustomerID": 0,
15        "Name": "",
16        "Email": "",
17        "Createdat": "0001-01-01T00:00:00Z",
18        "Accounts": null,
19        "Loans": null
20      }
21    }
22  }
```

B) SANTOSH REPAYS 2000 OF THAT AMOUNT



The screenshot shows the Postman interface with a POST request to `http://localhost:8080/loans/1/repay`. The request body is raw JSON:

```
1 {
2   "amount": 2000
3 }
4
```

The response status is `200 OK` with a response time of `29 ms` and a size of `791 B`. The response body is:

```
1   {
2     "success": true,
3     "data": {
4       "loan": {
5         "LoanID": 1,
6         "CustomerID": 1,
7         "BankID": 1,
8         "BranchID": 1,
9         "Principal": 10000,
10        "InterestRate": 12,
11        "RemainingAmount": 8000,
12        "Createdat": "2026-02-12T16:16:06.052062+05:30",
13        "Customer": {
14          "CustomerID": 0,
15          "Name": "",
16          "Email": "",
17          "Createdat": "0001-01-01T00:00:00Z",
18          "Accounts": null,
19          "Loans": null
20        }
21      }
22    }
23  }
```

C) VIEW LOAN DETAILS

The screenshot shows the Postman interface with a GET request to `http://localhost:8080/loans/1`. The response is a 200 OK status with a JSON payload. The JSON response is as follows:

```
1  {
2      "success": true,
3      "data": {
4          "interest_year": 960,
5          "loan": {
6              "LoanID": 1,
7              "CustomerID": 1,
8              "BankID": 1,
9              "BranchID": 1,
10             "Principal": 10000,
11             "InterestRate": 12,
12             "RemainingAmount": 8000,
13             "CreatedAt": "2026-02-12T16:16:06.052062+05:30",
14             "Customer": {
15                 "CustomerID": 0,
16                 "Name": "",
17                 "Email": "",
18                 "CreatedAt": "0001-01-01T00:00:00Z",
19                 "Accounts": null,
20                 "Loans": null
21             }
22         }
23     }
24 }
```

I separated:

- Account activity (transactions)
- Loan lifecycle (loan + repayment history)

This makes system scalable and normalized.

PRINCIPLE

To create this project I followed **SOLID PRINCIPLES ->**

- **S — Single Responsibility Principle (SRP):**

Here I designed 4 Layers and each layer only perform one task so only have 1 reason to change

Handler Layer → Service Layer → Repository Layer → Database

❖ *Handler Layer (API Layer)*

Purpose:

- Accept HTTP request
- Validate input
- Bind JSON
- Send response

No business logic here.

Because handlers should only deal with HTTP concerns.

❖ *Service Layer (Business Logic)*

This is the brain of the system.

Examples:

- Deposit validation
- Prevent negative withdrawal
- Check branch belongs to bank
- Check customer exists before opening account
- Apply 12% interest on loan creation
- Use transactions for atomic operations

It does NOT:

- Handle HTTP
- Write raw SQL

❖ ***Repository Layer (Database Access)***

Purpose:

- Only DB operations
- No business logic

Examples:

- Create()
- GetByID()
- UpdateBalance()

This makes system testable and clean.

❖ ***Database Layer***

PostgreSQL + GORM

- Auto migration
 - Foreign key support
 - ORM mapping
 - Clean struct-to-table mapping
-

- You change database → only repository changes
- You change business rules → only service changes
- You change API format → only handler changes

Thus there is separation of work.

● **O — Open/Closed Principle (OCP):**

I should be able to add new behavior without breaking old code.

OCP is followed in service layer in interfaces i created like in **account interface** We have **deposit(), withdraw()** , In future we can add new behaviors like **FreezeAccount()** etc and also can add various new account type like Current_account,joint_account, we can extend logic in service.

- **L — Liskov Substitution Principle (LSP):**

If a class implements an interface, it must behave correctly when substituted.

In my project:

Example:

```
type AccountRepository interface {  
    Create()  
    GetByID()  
}
```

If later I create:

MockAccountRepository

For testing.

It must behave same as real repository.

Your code already allows this because:

Services depend on interface, not concrete struct.

That satisfies LSP.

- **Interface Segregation Principle (ISP)**

Do not force a class to implement methods it does not use.

Example:

Instead of one huge repository:

BankingRepository

I created:

AccountRepository

LoanRepository

CustomerRepository

BranchRepository

Each interface is small and focused.

Services only depend on what they need.

Example:

AccountService depends only on:

- AccountRepository
- TransactionRepository
- CustomerRepository

It does NOT depend on LoanRepository.

- **D — Dependency Inversion Principle (DIP)**

High-level modules should not depend on low-level modules.

Both should depend on abstractions.

*In My Project AccountService
depends on
AccountRepository (interface)*

Then in main.go:

```
accountRepo := repositories.NewAccountRepository(dbConn)
accountService := services.NewAccountService(accountRepo, ...)
```

Instead of direct **Account Service using GORM** there is layer of abstraction called Repository(Interface)

I followed SOLID principles to ensure maintainability and scalability. I separated concerns using a layered architecture where handlers manage HTTP, services manage business rules, and repositories manage persistence. All services depend on abstractions rather than concrete implementations, allowing future extension without modifying core logic. This ensures clean code, easier testing, and better scalability.