P2PLoop: A Peer to Peer Lending System

By Shyam Sankar(335006501)

A peer-to-peer lending system is a system that bypasses traditional systems like banks and allows customers to lend and borrow money from each other. The system would allow lending users to invest money towards a pool, where from borrowing customers can take loans. This is an innovative approach that has come into prominence with the rise of decentralized finance.

Benefits of P2P Banking

- **Lower Costs**: Compared to banks, a peer to peer lending system would often have lower interest rates on loans and higher returns to lenders.
- **Accessibility**: Users with limited credit history and small business in need for a quick loan might benefit from using a P2P system.
- **Convenience**: While taking a loan from a bank might entail a long waiting time, the P2P banking system would aim for quick loan dispersal owing to a fully digitized and automated process.
- **Transparency**: Lenders can see exactly where their money is going and choose investments based on personal criteria, while borrowers get clear terms upfront.
- **Decentralization**:Fosters a more decentralized approach to lending money without any reliance on established financial institutions or middle-men.

I believe the peer-to-peer (P2P) banking system represents the future of financial services, driven by technology to create a more efficient, inclusive, and autonomous way of lending and borrowing money. Traditional banks have long been the intermediaries between borrowers and lenders, often imposing high fees, interest rates, and lengthy approval processes. P2P systems, however, allow individuals to connect directly, cutting out these intermediaries and enabling faster, more flexible transactions. This efficiency is crucial in today's fast-paced world, as borrowers can quickly obtain loans, and lenders can earn returns with minimal friction.

Furthermore, P2P banking helps address financial **inclusivity**. Many individuals, particularly in underserved communities, are often excluded from traditional banking due to rigid credit requirements. P2P platforms open up access to loans for a broader range of people, offering a more personalized and accessible alternative.

Moreover, P2P banking empowers both lenders and borrowers with **autonomy** over their transactions. Lenders can choose who they wish to lend to, and borrowers can often find better terms than those offered by traditional banks. This flexibility creates a more tailored and transparent lending environment, where users have more control over their financial decisions. As technology continues to evolve, the integration of AI, blockchain, and other advancements will further enhance P2P systems, making them even more secure and efficient. Ultimately, P2P banking has the potential to transform the financial landscape, offering a future where money is lent and borrowed in a more streamlined and equitable manner.

Part (a) Application Description:

The application will allow an administrator to derive insights from the data present in the database, allow for the visualization of relevant insights and perform operations. The end user for whom the system is designed is an admin who works at P2PLoop and monitors activity and helps customers.

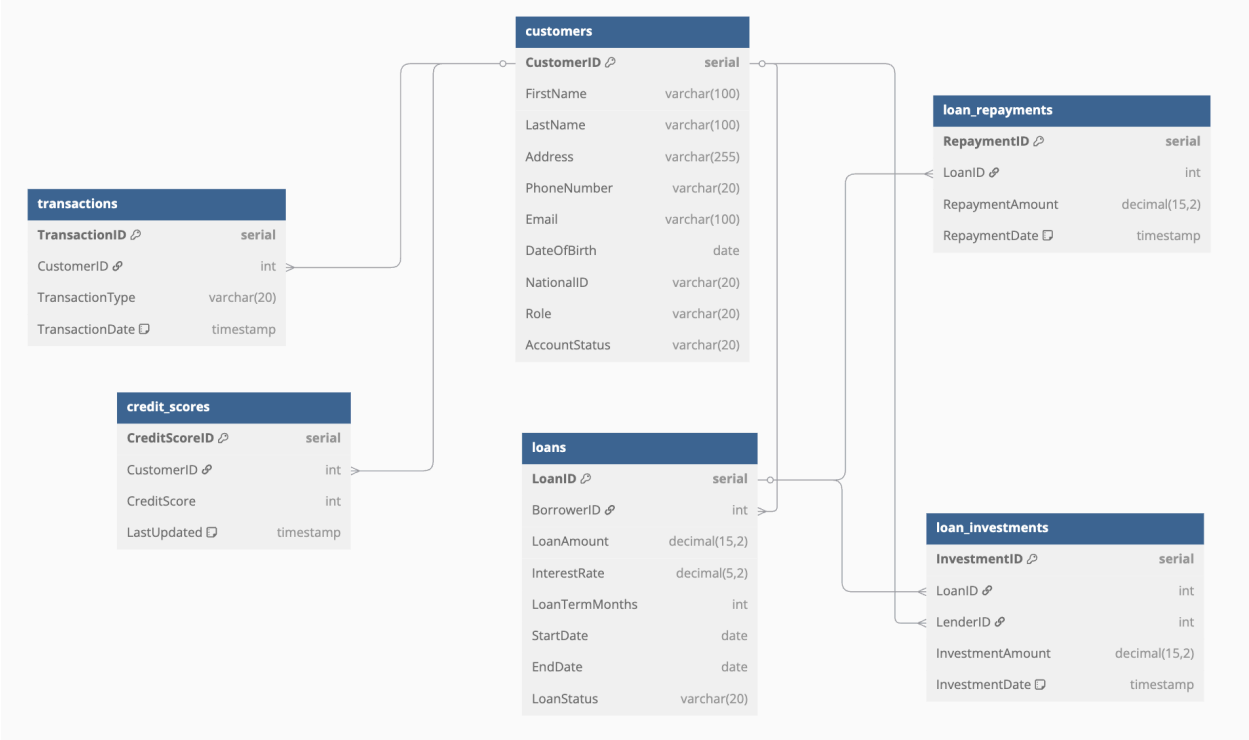
The application will have three sections accessible through the navigation bar:

1. The **landing page** will have visualizations of relevant trends derived from the tables along with a searching functionality that will allow the user to search by given fields to enquire on the corresponding customers.
2. The **customers page** will allow the user to add new customers and will feature a table containing details on all the customers
3. The **loans page** will allow the user to view details by searching for loans and will feature methods to make payments.

For further details on each functionality and interactive elements within the application, go to Part(f).

Full Source Code: [Github/P2PLoop]

Part (b) E/R Diagram:



Further details on each relation is available in the following section.

Part (c) Database Schema & Normalization

1. Customers Table

1. **Primary Key**: CustomerID
2. **Purpose**: Stores customer information.
3. **Relationships**:
    a. **Referenced by Loans**: The BorrowerID column in Loans is a foreign key referencing Customers(CustomerID). This establishes a **one-to-many relationship**: one customer (as a borrower) can have multiple loans.
    b. **Referenced by Transactions**: The CustomerID column in Transactions is a foreign key referencing Customers(CustomerID). This is a **one-to-many relationship**: one customer can have multiple transactions.
    c. **Referenced by CreditScores**: The CustomerID column in CreditScores is a foreign key referencing Customers(CustomerID). This is a **one-to-many relationship**: one customer can have multiple credit score records (e.g., over time as indicated by the "lastupdated" column).
    d. **Referenced by LoanInvestments**: The LenderID column in LoanInvestments is a foreign key referencing Customers(CustomerID). This is a **one-to-many relationship**: one customer (as a lender) can invest in multiple loans.

Non Trivial Functional Dependencies:

1. **CustomerID → FirstName, LastName, Address, PhoneNumber, Email, DateOfBirth, NationalID, Role, AccountStatus**
    a. The primary key CustomerID uniquely identifies a customer, so it determines all other attributes in the table.
2. **NationalID → CustomerID, FirstName, LastName, Address, PhoneNumber, Email, DateOfBirth, Role, AccountStatus**
    a. Since NationalID is marked as UNIQUE, it uniquely identifies a customer in the real world (e.g., a government-issued ID). Thus, it determines all other attributes, including CustomerID.

2. Loans Table

1. **Primary Key**: LoanID
2. **Purpose**: Tracks loan details for borrowers.
3. **Relationships**:
    a. **References Customers**: The BorrowerID column is a foreign key referencing Customers(CustomerID). This is a **many-to-one relationship**: many loans can belong to a single borrower (customer).
    b. **Referenced by LoanRepayments**: The LoanID column in LoanRepayments is a foreign key referencing Loans(LoanID). This is a **one-to-many relationship**: one loan can have multiple repayments.
    c. **Referenced by LoanInvestments**: The LoanID column in LoanInvestments is a foreign key referencing Loans(LoanID). This is a **one-to-many relationship**: one loan can have multiple investments from different lenders.

Non Trivial Functional Dependencies:

1. **LoanID → BorrowerID, LoanAmount, InterestRate, LoanTermMonths, StartDate, EndDate, LoanStatus**
    a. The primary key LoanID uniquely identifies a loan, so it determines all other attributes in the table.

3. LoanRepayments Table

1. **Primary Key**: RepaymentID
2. **Purpose**: Records repayment details for loans.
3. **Relationships**:
    a. **References Loans**: The LoanID column is a foreign key referencing Loans(LoanID). This is a **many-to-one relationship**: many repayments can be associated with a single loan.

Non Trivial Functional Dependencies:

1. **RepaymentID → LoanID, RepaymentAmount, RepaymentDate**
    a. The primary key RepaymentID uniquely identifies a repayment, so it determines all other attributes.

4. Transactions Table

1. **Primary Key**: TransactionID
2. **Purpose**: Logs financial transactions for customers.
3. **Relationships**:
    a. **References Customers**: The CustomerID column is a foreign key referencing Customers(CustomerID). This is a **many-to-one relationship**: many transactions can be associated with a single customer.

Non Trivial Functional Dependencies:

1. **TransactionID → CustomerID, TransactionType, Amount, TransactionDate**
    a. The primary key TransactionID uniquely identifies a transaction, so it determines all other attributes.

5. CreditScores Table

1. **Primary Key**: CreditScoreID
2. **Purpose**: Tracks credit scores for customers over time.
3. **Relationships**:
    a. **References Customers**: The CustomerID column is a foreign key referencing Customers(CustomerID). This is a **many-to-one relationship**: many credit score records can belong to a single customer.

Non Trivial Functional Dependencies:

1. **CreditScoreID → CustomerID, CreditScore, LastUpdated**
    a. The primary key CreditScoreID uniquely identifies a credit score record, so it determines all other attributes.

6. LoanInvestments Table

1. **Primary Key**: InvestmentID
2. **Purpose**: Records investments made by lenders in specific loans.
3. **Relationships**:
    a. **References Loans**: The LoanID column is a foreign key referencing Loans(LoanID). This is a **many-to-one relationship**: many investments can be made in a single loan.
    b. **References Customers**: The LenderID column is a foreign key referencing Customers(CustomerID). This is a **many-to-one relationship**: many investments can be made by a single lender (customer).

Non Trivial Functional Dependencies:

1. **InvestmentID → LoanID, LenderID, InvestmentAmount, InvestmentDate**
    a. The primary key InvestmentID uniquely identifies an investment, so it determines all other attributes

To perform Boyce-Codd normal form check and necessary decomposition

1. Customers Table

Functional Dependencies:

1. CustomerID → FirstName, LastName, Address, PhoneNumber, Email, DateOfBirth, NationalID, Role, AccountStatus
2. NationalID → CustomerID, FirstName, LastName, Address, PhoneNumber, Email, DateOfBirth, Role, AccountStatus

Not in BCNF due to NationalID not being a superkey.

We can decompose the relation to get the following relations:

1. **Customers_NationalID**: (NationalID (PK), CustomerID (FK REFERENCES Customers(CustomerID)))
2. **Customers**: (CustomerID (PK), FirstName, LastName, Address, PhoneNumber, Email, DateOfBirth, Role, AccountStatus)

2. Loans Table

LoanID → BorrowerID, LoanAmount, InterestRate, LoanTermMonths, StartDate, EndDate, LoanStatus

LoanID is the primary key (a superkey). **This FD satisfies BCNF.**

3. LoanRepayments Table

RepaymentID → LoanID, RepaymentAmount, RepaymentDate

RepaymentID is the primary key (a superkey). **This FD satisfies BCNF.**

4. Transactions Table

TransactionID → CustomerID, TransactionType, Amount, TransactionDate

TransactionID is the primary key (a superkey). **This FD satisfies BCNF.**

5. CreditScores Table

CreditScoreID → CustomerID, CreditScore, LastUpdated

CreditScoreID is the primary key (a superkey). **This FD satisfies BCNF.**

6. LoanInvestments Table

InvestmentID → LoanID, LenderID, InvestmentAmount, InvestmentDate

InvestmentID is the primary key (a superkey)**. This FD satisfies BCNF.**

Part (d) Tables & Data:

The database was created using **PostgreSQL**. The schema will feature 6 tables. After decomposition for BCNF, the 7th table was added.

Queries used to created the tables within the database: Creating the database

Python code to populate the tables: db_populate.py

1. Customers Table: 10000 entries.
2. Loans Table: 500 entries
3. LoanRepayments Table: 5000 entries
4. Transactions Table: 5000 entries
5. Credit Scores Table: 10000 entries
6. LoanInvestments Table: 5000 entries
7. CustomersNationalID: 10000 entries

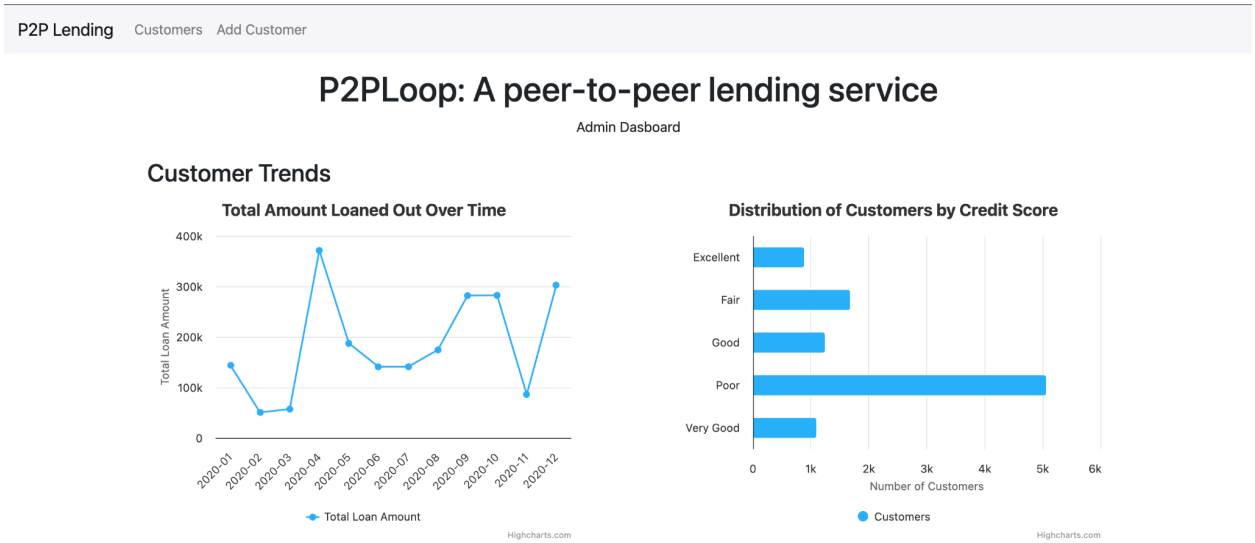
Details on the columns in each table, keys and functional dependencies are available above in the ER diagram and the description of each table following it.

Part (e) User Interface & Functions

The P2P Lending System uses the following technologies:

1. **Frontend**: The user interface is built using **HTML**, **CSS**, and **JavaScript**, with the addition of libraries: **Bootstrap** for responsive design and **Highcharts** for data visualization.
2. **Backend**: The server-side logic is implemented using **Flask**. It handles user requests, manages the database, and processes financial transactions.

The application has three pages with distinct functionalities as described below



Customer trends involves 4 visuals:

1. **Total Amount Loaned Over Time:**

```
SELECT TO_CHAR(startdate, 'YYYY-MM') AS Month, sum(loanamount) AS LoanedAmount
  FROM Loans
  GROUP BY TO_CHAR(startdate, 'YYYY-MM')
  ORDER BY Month;
```

2. **Distribution of Customers by Credit Score**

```
SELECT TO_CHAR(repaymentdate, 'YYYY-MM-DD') AS Month, sum(repaymentamount) AS
RepaidAmount
  FROM Loanrepayments
  GROUP BY TO_CHAR(repaymentdate, 'YYYY-MM-DD')
  LIMIT 7;
```

Top Borrowing Customers

Customer ID	Customer Name	Total Borrowed
4961	Karina Silva	72643.01
8534	Ashley Johnson	62645.06
3084	Willie White	58061.59
3182	Rachel Hancock	50700.50
2406	Robert Johnson	50651.73

Top Lenders

Customer ID	Customer Name	Total Lended
266	Nancy Price	31393.00
9794	Katherine Lyons	29406.53
3834	John Burgess	29076.91
9697	Steven Kim	27183.48
5453	Philip Sanchez	27129.23

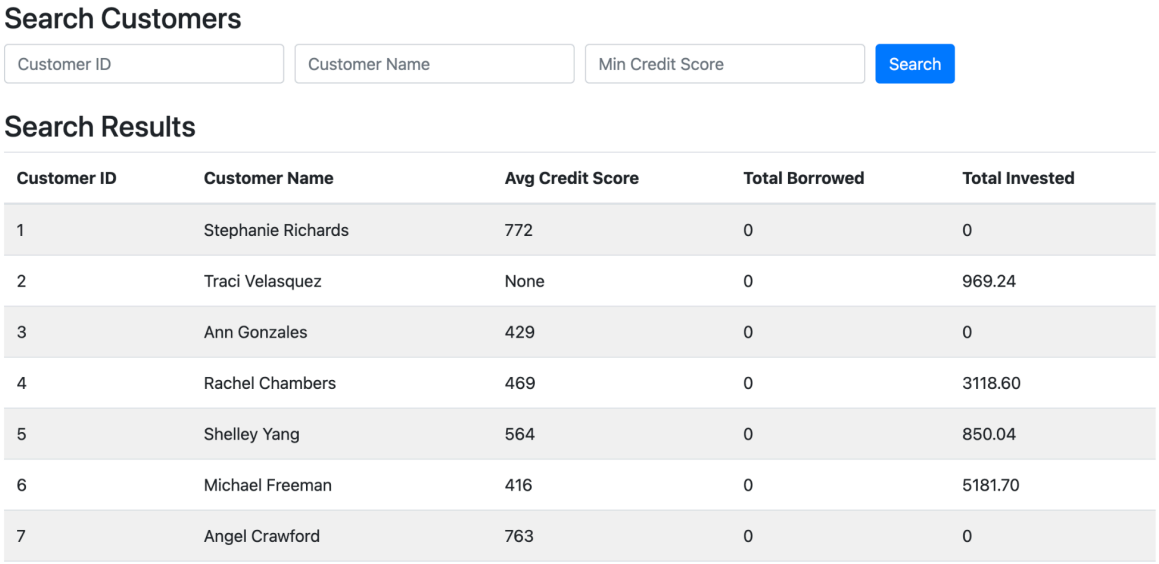
3. Top Borrowing Customers

```
SELECT
   c.CustomerID,
   c.FirstName || ' ' || c.LastName AS CustomerName,
   SUM(l.LoanAmount) AS TotalBorrowed
FROM Customers c
JOIN Loans l ON c.CustomerID = l.BorrowerID
GROUP BY c.CustomerID, c.FirstName, c.LastName
ORDER BY TotalBorrowed DESC
LIMIT 5;
```

4. Top Lending Customers:

```
SELECT
   c.CustomerID,
   c.FirstName || ' ' || c.LastName AS CustomerName,
   SUM(li.InvestmentAmount) AS TotalInvested
FROM Customers c
JOIN LoanInvestments li ON c.CustomerID = li.LenderID
GROUP BY c.CustomerID, c.FirstName, c.LastName
ORDER BY TotalInvested DESC
LIMIT 5;
```

5. Search Customers



A searching functionality has been implemented for the admin to search customers by ID, name or minimum credit score.

```
SELECT
    c.CustomerID,
    c.FirstName || ' ' || c.LastName AS CustomerName,
                round(avg(creditscore)) as creditscores,
    COALESCE(SUM(l.loanamount), 0) AS TotalBorrowed,
    COALESCE(SUM(t.investmentamount), 0) AS TotalInvested
FROM
    Customers c
LEFT JOIN CreditScores cs ON c.CustomerID = cs.CustomerID
LEFT JOIN Loans l ON l.BorrowerID = c.CustomerID
LEFT JOIN loaninvestments t ON t.LenderID = c.CustomerID
GROUP BY
    c.CustomerID, c.FirstName, c.LastName
ORDER BY
    c.CustomerID;
```

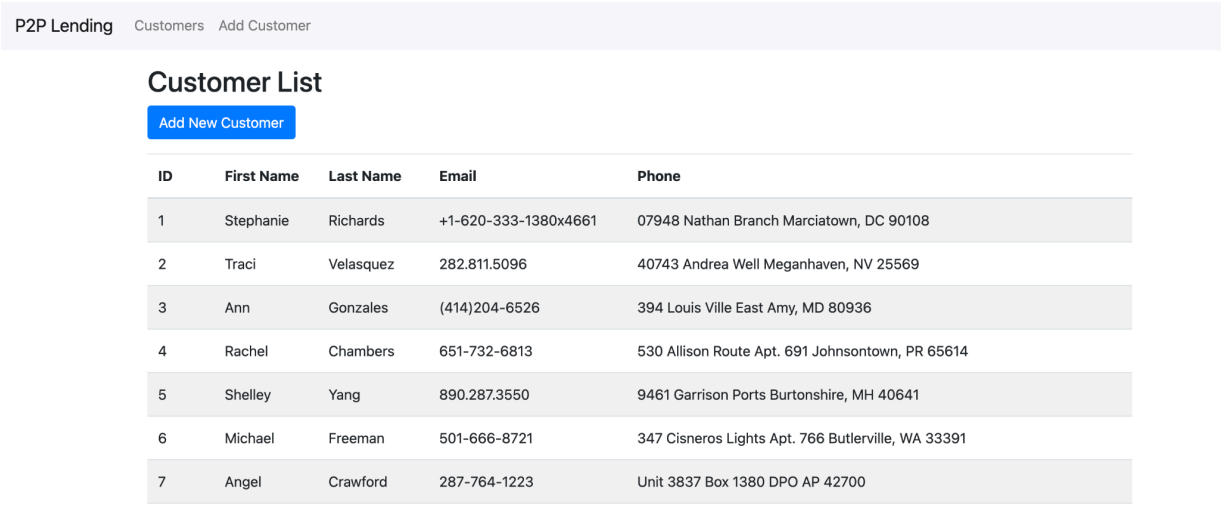
Conditional:

```
AND c.CustomerID = %s
AND (c.FirstName || ' ' || c.LastName) LIKE %s
AND cs.creditscore >= %s
```

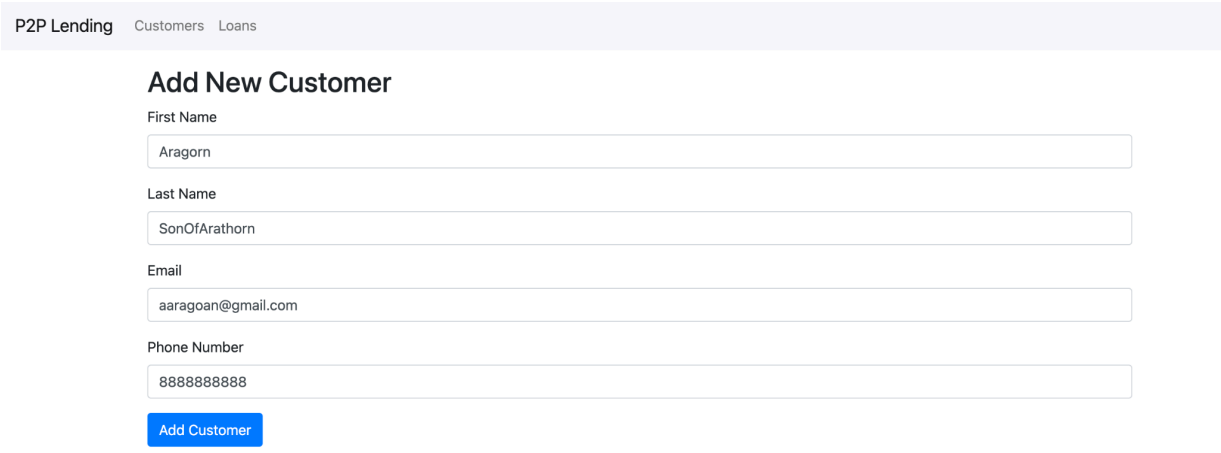
5. Add Customers

Functionality allows for enquiring details on different customers and adding new customers

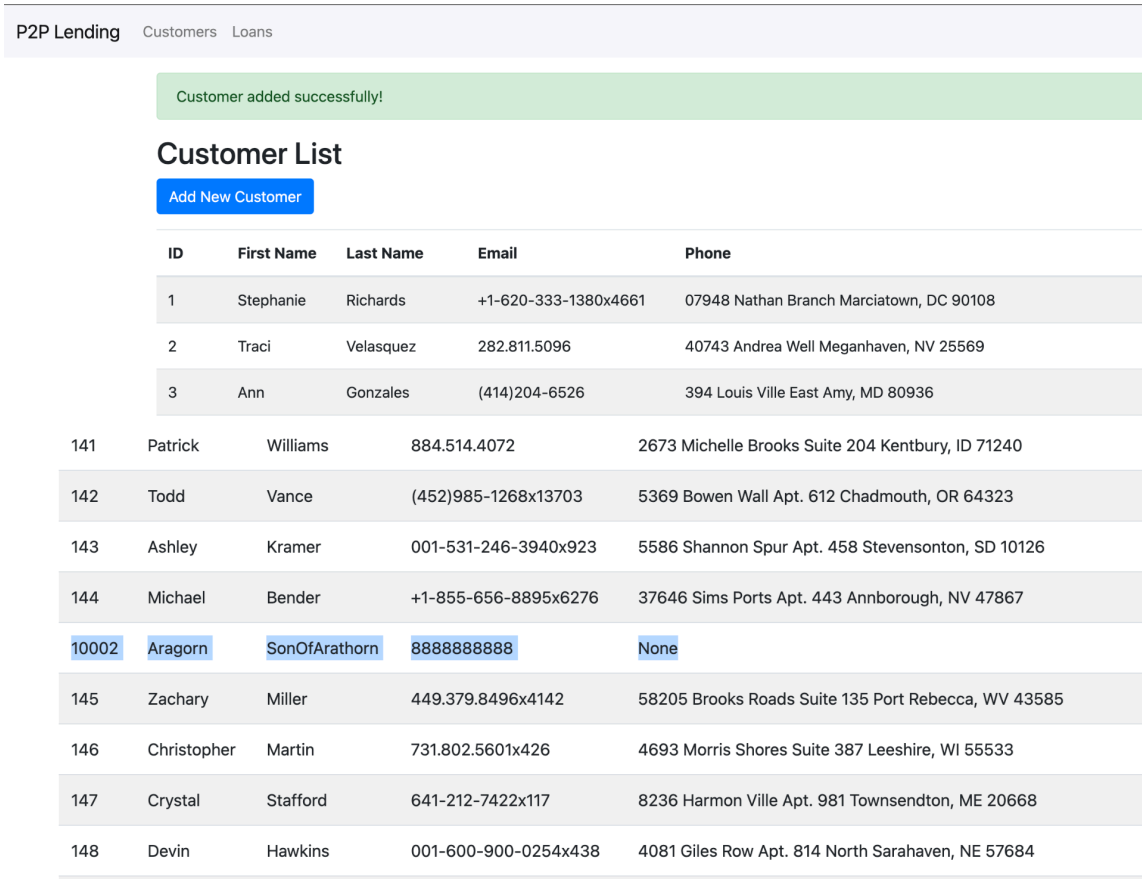
1. Go to Customers tab
2. Click on “Add New Customer”



3. Enter details



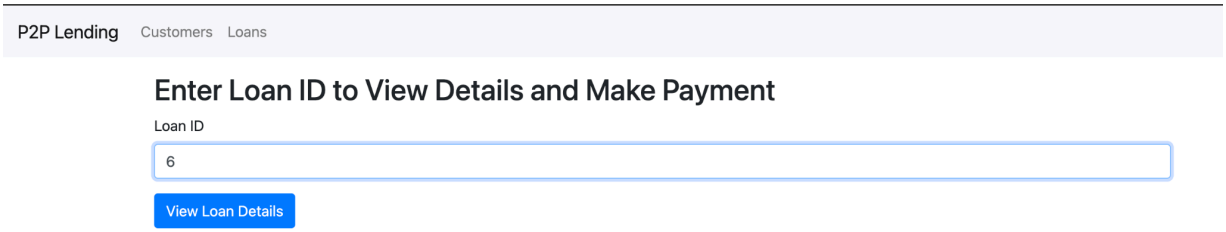
4. Click enter and find your details in the table



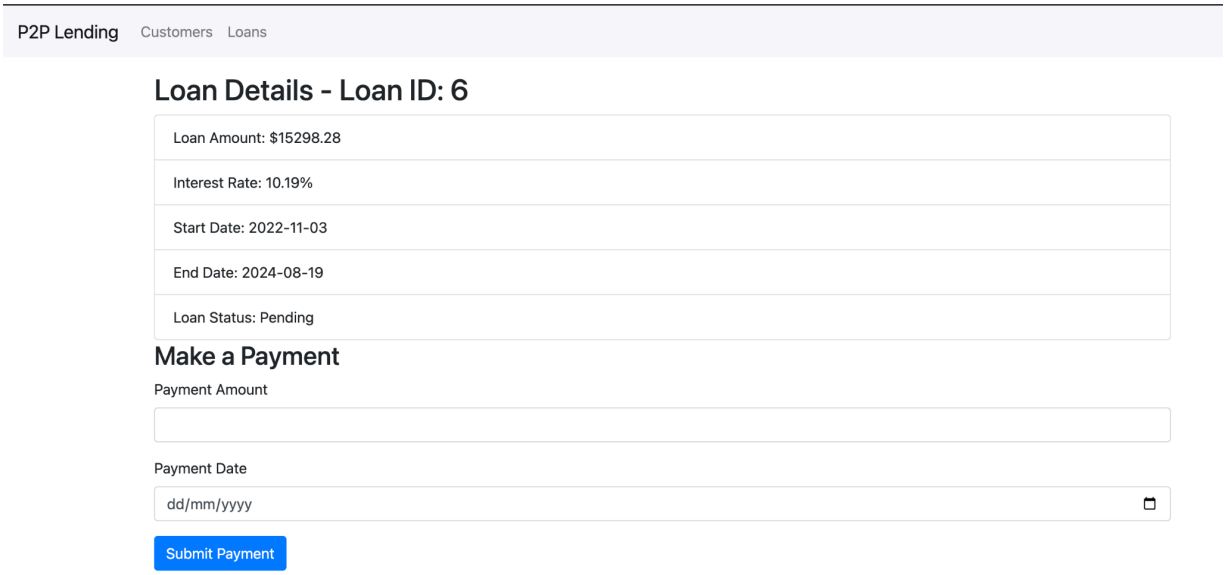
6. Loans & Payment

Functionality allows for viewing loan details and making payments towards loans.

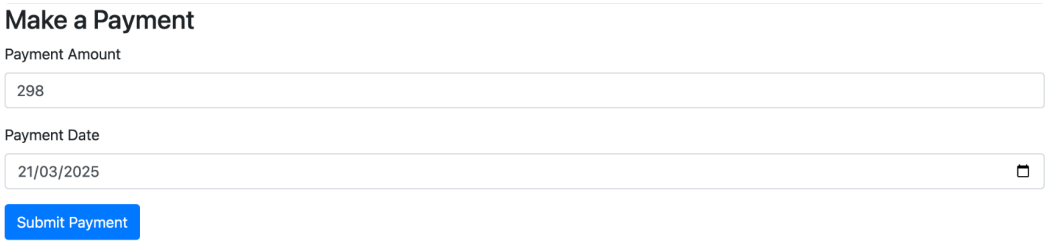
1. Go to the Loans tab
2. Enter loan id to see loan details.



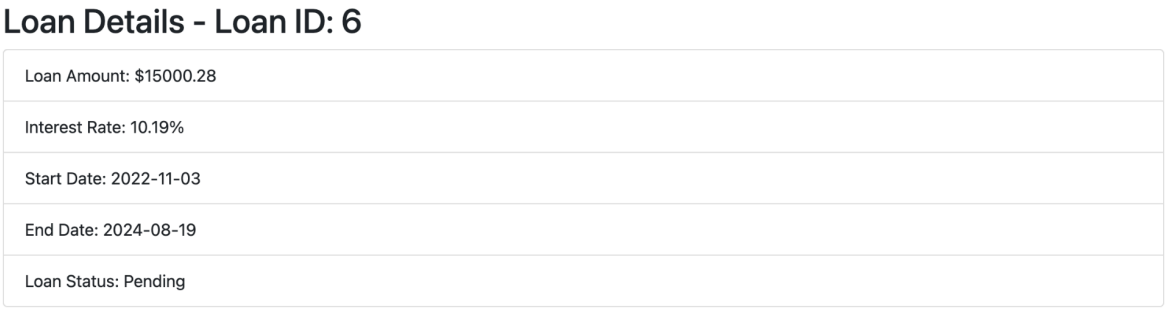
3. Click on the button to see details



4. Make a payment



5. Amount Updated



Loan status will change to “completed” if the full amount is paid.

Part (f) Discussion

In developing a database application, particularly for a P2P loan system, the process involved creating a web application with Flask, designing the necessary database schema, integrating user interactions, and managing transactions and loan repayments. The backend database (PostgreSQL) stored essential information about the loans, repayments, and borrowers, while the frontend allowed users to interact with the system seamlessly.

This was my first time using Flask and PostgreSQL, and it was a great learning experience. While I was already familiar with basic HTML and JavaScript, diving into Flask for the backend and PostgreSQL for the database management was a new challenge. Initially, I had to get comfortable with setting up Flask routes, and managing server-side logic. Learning how to integrate PostgreSQL for data storage and retrieval was also a significant step, especially when it came to writing SQL queries and managing database connections. While it took some time to get everything set up, the project helped me gain a solid understanding of backend development and how to connect a web application to a relational database. The combination of learning Flask and PostgreSQL, along with my existing knowledge of HTML and JavaScript, gave me a much broader understanding of full-stack development.