CS-GY 6083 – Section B, Spring 2024

Principles of Database Systems

NEW YORK UNIVERSITY
Tandon School of Engineering

Project Part II

Date of submission: May 5th, 2024

Team Members:

Anant Dayal
ad6829

Chandana Thimmalapura Jagadeeshaiah
ct3002

Gyanesh Ravishankar Pandey
grp7887

TABLE OF CONTENTS

Summary

Business Case:

SAFE Bank, short for Save And Fortune Excellence, manages customer data and various account types including Checking, Savings, and Loan. Each account type possesses specific attributes tailored to its function, such as service charges for Checking, interest rates for Savings, and loan details for Loans. Moreover, the bank has outlined specific business rules to streamline operations and enhance customer service by implementing a robust database.

Alongside the database schema development, a pivotal aspect of the project involved creating a web-based user interface to facilitate customer and employee interactions with the database.

Approach Towards Business Solution:

To address SAFE Bank's requirements, we designed a comprehensive database schema capable of efficiently storing and managing customer information and account details. Employing relational database principles, we structured tables for Customers, Checking Accounts, Savings Accounts, and Loans, ensuring each entity maintains its unique attributes while facilitating seamless data retrieval and manipulation.

For instance, we created separate tables for different account types to uphold the rule against multiple accounts of the same type for a single customer. Furthermore, to accommodate the specific requirements of student loans and home loans, we extended the schema to include additional tables for Student Loan Details and Home Loan Details. This facilitated the storage of relevant information such as educational institute details for student loans and house-built year and insurance details for home loans.

We developed a comprehensive web application using Django, a high-level Python web framework known for its scalability and rapid development capabilities. Leveraging Django's built-in features, we designed a user-friendly interface that allows customers or admin users to register, log in, and perform essential business activities such as creating customers/accounts, reading the transaction details, updating the password and being able to delete the accounts (CRUD operations).

The web application interfaces seamlessly with the MySQL database, storing and retrieving customer details, account information, and transaction records.

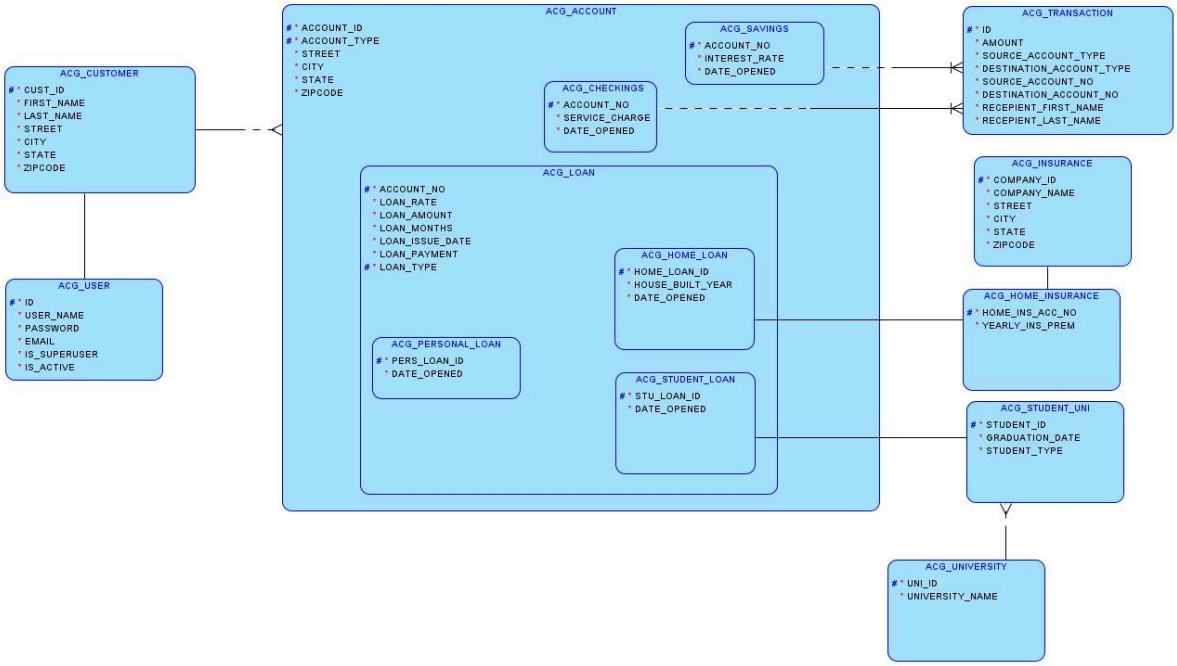
Improving Business Performance:

By implementing this integrated solution, SAFE Bank gains significant advantages in terms of efficiency, customer service, and data management. The web-based interface simplifies customer interactions, enabling seamless account creation, transaction processing, and account

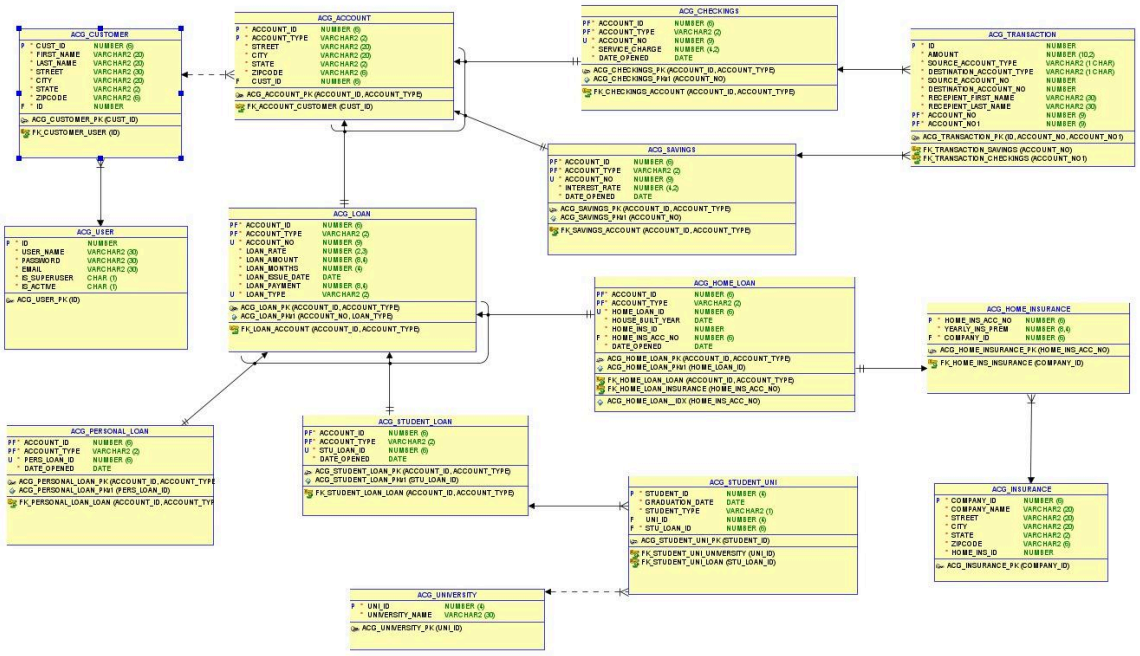
management. Customers can register their details, open bank accounts, and conduct transactions with ease, enhancing their overall banking experience.

Overall, the combination of Django for web application development and MySQL for database management offers SAFE Bank a scalable and reliable solution to meet its business needs effectively. By streamlining operations and improving customer engagement, this solution contributes to enhanced business performance and customer satisfaction.

Logical Model



Relational Model



Assumptions made in support of designs.

- Account ID is unique for each customer and acts as the master bank account ID.
- Every type of account is unique, and hence the combination of account ID and account type determines uniqueness.
- The date opened will be different for different types of accounts.
- loan_purpose is a newly added attribute for personal loan type which describes the purpose for which the loan was taken, e.g., medical expenses, home improvement, etc.
- Graduation month and year are stored as datatype - DATE (YYYY - MM - DD).
- house_built_year is stored as a date data type which provides flexibility in querying and analyzing the data.
- It is compulsory to have insurance for a home loan.
- A customer can take only one loan at a time as a customer cannot have multiple accounts of the same type according to the business rule.
- The relationship between a university and its students is one-to-many, as multiple students from the same university may have taken out loans.

Brief details of software, programming language, and database used.

For the development of the web application, we used a combination of technologies to ensure efficiency, scalability, and reliability.

Software: We used Django as the primary software, providing a robust framework for web development. Django, built on Python, offered a comprehensive suite of tools and conventions that significantly expedited our development process while ensuring the security and scalability required for a banking application.

Programming Language: Python served as the programming language for implementing the business logic and backend functionality of the web application.

Front-end: For the front-end, we utilized HTML, CSS, and JavaScript, with Tailwind CSS enhancing our styling and layout capabilities. HTML provided the structure, CSS leveraged Tailwind's utility classes to streamline styling, while JavaScript added interactivity and dynamic behavior to the web pages.

Code Editor: Visual Studio Code was our preferred code editor for writing and editing code.

Database: We chose MySQL as the database management system (DBMS) for storing and managing the structured data required for the web application. MySQL offered reliability, performance, and scalability, making it an ideal choice for handling customer information, account details, and transaction records.

Design: For designing the database schema, we relied on Oracle Data Modeler. Its intuitive interface and powerful features facilitated the design process.

MVC Architecture: In our banking application, we embraced the Model-View-Controller (MVC) architecture, prioritizing the separation of concerns. By partitioning our codebase into models for data handling, views for user interface presentation, and controllers for application logic, we achieved robust code organization and maintainability, facilitating efficient collaboration among developers.

Data Visualization: Matplotlib was employed for data visualization, enabling us to create insightful graphs and charts to present data effectively. Specifically, Matplotlib was used to show admin analytics in Pie chart form enhancing our ability to convey complex information visually.

By leveraging these technologies and tools, we created a robust and efficient web application that meets the needs of SAFE Bank.

DDL file content, generated from Data Modeler Relational Model

DDL CODE IN Oracle

```
-- Generated by Oracle SQL Developer Data Modeler 23.1.0.087.0806
--   at:        2024-05-05 22:42:24 GMT-04:00
--   site:      Oracle Database 21c
--   type:      Oracle Database 21c



-- predefined type, no DDL - MDSYS.SDO_GEOMETRY

-- predefined type, no DDL - XMLTYPE

CREATE TABLE acg_account (
    account_id   NUMBER(6) NOT NULL,
    account_type VARCHAR2(2) NOT NULL,
    street       VARCHAR2(20) NOT NULL,
    city         VARCHAR2(20) NOT NULL,
    state        VARCHAR2(2) NOT NULL,
    zipcode      VARCHAR2(6) NOT NULL,
    cust_id      NUMBER(6)
);
```

```
ALTER TABLE acg_account
   ADD CONSTRAINT ch_inh_acg_account CHECK ( account_type IN ( 'C','L','S') );

COMMENT ON COLUMN acg_account.account_id IS
   'Unique identifier for a customer with an account in the bank';

ALTER TABLE acg_account ADD CONSTRAINT acg_account_pk PRIMARY KEY ( account_id,
                                                                    account_type );

CREATE TABLE acg_checkings (
   account_id     NUMBER(6) NOT NULL,
   account_type   VARCHAR2(2) NOT NULL,
   account_no     NUMBER(9) NOT NULL,
   service_charge NUMBER(4, 2) NOT NULL,
   date_opened    DATE NOT NULL
);

COMMENT ON COLUMN acg_checkings.account_id IS
   'Unique identifier for a customer with an account in the bank';

ALTER TABLE acg_checkings ADD CONSTRAINT acg_checkings_pk PRIMARY KEY ( account_id,
                                                                       account_type );
```

```
ALTER TABLE acg_checkings ADD CONSTRAINT acg_checkings_pkv1 UNIQUE ( account_no );


CREATE TABLE acg_customer (
    cust_id    NUMBER(6) NOT NULL,
    first_name VARCHAR2(20) NOT NULL,
    last_name  VARCHAR2(20) NOT NULL,
    street     VARCHAR2(30) NOT NULL,
    city       VARCHAR2(20) NOT NULL,
    state      VARCHAR2(2) NOT NULL,
    zipcode    VARCHAR2(6) NOT NULL,
    id         NUMBER NOT NULL
);


ALTER TABLE acg_customer ADD CONSTRAINT acg_customer_pk PRIMARY KEY ( cust_id );


CREATE TABLE acg_home_insurance (
    home_ins_acc_no NUMBER(6) NOT NULL,
    yearly_ins_prem NUMBER(8, 4) NOT NULL,
    company_id      NUMBER(6) NOT NULL
);
```

```
ALTER TABLE acg_home_insurance ADD CONSTRAINT acg_home_insurance_pk
PRIMARY KEY ( home_ins_acc_no );

CREATE TABLE acg_home_loan (
    account_id       NUMBER(6) NOT NULL,
    account_type     VARCHAR2(2) NOT NULL,
    home_loan_id     NUMBER(6) NOT NULL,
    house_built_year DATE NOT NULL,
    home_ins_id      NUMBER NOT NULL,
    home_ins_acc_no  NUMBER(6) NOT NULL,
    date_opened      DATE NOT NULL
);

COMMENT ON COLUMN acg_home_loan.account_id IS
    'Unique identifier for a customer with an account in the bank';

CREATE UNIQUE INDEX acg_home_loan__idx ON
    acg_home_loan (
        home_ins_acc_no
    ASC );

ALTER TABLE acg_home_loan ADD CONSTRAINT acg_home_loan_pk PRIMARY KEY (
account_id,
                                          account_type );
```

```
ALTER TABLE acg_home_loan ADD CONSTRAINT acg_home_loan_pkv1 UNIQUE ( home_loan_id );


CREATE TABLE acg_insurance (
    company_id   NUMBER(6) NOT NULL,
    company_name VARCHAR2(20) NOT NULL,
    street       VARCHAR2(20) NOT NULL,
    city         VARCHAR2(20) NOT NULL,
    state        VARCHAR2(2) NOT NULL,
    zipcode      VARCHAR2(6) NOT NULL,
    home_ins_id  NUMBER NOT NULL
);


ALTER TABLE acg_insurance ADD CONSTRAINT acg_insurance_pk PRIMARY KEY ( company_id );


CREATE TABLE acg_loan (
    account_id      NUMBER(6) NOT NULL,
    account_type    VARCHAR2(2) NOT NULL,
    account_no      NUMBER(9) NOT NULL,
    loan_rate       NUMBER(2, 3) NOT NULL,
    loan_amount     NUMBER(8, 4) NOT NULL,
    loan_months     NUMBER(4) NOT NULL,
```

```
    loan_issue_date DATE NOT NULL,
    loan_payment    NUMBER(8, 4) NOT NULL,
    loan_type       VARCHAR2(2) NOT NULL
);

ALTER TABLE acg_loan
    ADD CONSTRAINT ch_inh_acg_loan CHECK ( loan_type IN ( 'HL', 'PL', 'SL' ) );

COMMENT ON COLUMN acg_loan.account_id IS
    'Unique identifier for a customer with an account in the bank';

ALTER TABLE acg_loan ADD CONSTRAINT acg_loan_pk PRIMARY KEY ( account_id,
                                                              account_type );

ALTER TABLE acg_loan ADD CONSTRAINT acg_loan_pkv1 UNIQUE ( account_no,
                                                           loan_type );

CREATE TABLE acg_personal_loan (
    account_id   NUMBER(6) NOT NULL,
    account_type VARCHAR2(2) NOT NULL,
    pers_loan_id NUMBER(6) NOT NULL,
    date_opened  DATE NOT NULL
);
```

```
COMMENT ON COLUMN acg_personal_loan.account_id IS
    'Unique identifier for a customer with an account in the bank';

ALTER TABLE acg_personal_loan ADD CONSTRAINT acg_personal_loan_pk PRIMARY KEY ( account_id,
                                                                                account_type );

ALTER TABLE acg_personal_loan ADD CONSTRAINT acg_personal_loan_pkv1 UNIQUE ( pers_loan_id );

CREATE TABLE acg_savings (
    account_id    NUMBER(6) NOT NULL,
    account_type  VARCHAR2(2) NOT NULL,
    account_no    NUMBER(9) NOT NULL,
    interest_rate NUMBER(4, 2) NOT NULL,
    date_opened   DATE NOT NULL
);

COMMENT ON COLUMN acg_savings.account_id IS
    'Unique identifier for a customer with an account in the bank';

ALTER TABLE acg_savings ADD CONSTRAINT acg_savings_pk PRIMARY KEY ( account_id,
                                                                    account_type );
```

```
ALTER TABLE acg_savings ADD CONSTRAINT acg_savings_pkv1 UNIQUE ( account_no );

CREATE TABLE acg_student_loan (
    account_id   NUMBER(6) NOT NULL,
    account_type VARCHAR2(2) NOT NULL,
    stu_loan_id  NUMBER(6) NOT NULL,
    date_opened  DATE NOT NULL
);

COMMENT ON COLUMN acg_student_loan.account_id IS
    'Unique identifier for a customer with an account in the bank';

ALTER TABLE acg_student_loan ADD CONSTRAINT acg_student_loan_pk PRIMARY KEY
( account_id,
                                                account_type );

ALTER TABLE acg_student_loan ADD CONSTRAINT acg_student_loan_pkv1 UNIQUE (
stu_loan_id );

CREATE TABLE acg_student_uni (
    student_id      NUMBER(4) NOT NULL,
    graduation_date DATE NOT NULL,
    student_type    VARCHAR2(1) NOT NULL,
```

```
    uni_id          NUMBER(4),
    stu_loan_id     NUMBER(6) NOT NULL
);

ALTER TABLE acg_student_uni ADD CONSTRAINT acg_student_uni_pk PRIMARY KEY ( student_id );

CREATE TABLE acg_transaction (
    id                      NUMBER NOT NULL,
    amount                  NUMBER(10, 2) NOT NULL,
    source_account_type     VARCHAR2(1 CHAR) NOT NULL,
    destination_account_type VARCHAR2(1 CHAR) NOT NULL,
    source_account_no       NUMBER NOT NULL,
    destination_account_no  NUMBER NOT NULL,
    recepient_first_name    VARCHAR2(30) NOT NULL,
    recepient_last_name     VARCHAR2(30) NOT NULL,
    account_no              NUMBER(9) NOT NULL,
    account_no1             NUMBER(9) NOT NULL
);

COMMENT ON COLUMN acg_transaction.amount IS
    'amount transferred';

COMMENT ON COLUMN acg_transaction.source_account_type IS
```

```
    'source account type';

COMMENT ON COLUMN acg_transaction.destination_account_type IS
    'destination account type';

COMMENT ON COLUMN acg_transaction.source_account_no IS
    'source account number';

COMMENT ON COLUMN acg_transaction.destination_account_no IS
    'destination account number';

COMMENT ON COLUMN acg_transaction.recepient_first_name IS
    'first name of the recepient';

COMMENT ON COLUMN acg_transaction.recepient_last_name IS
    'last name of the recepient ';

ALTER TABLE acg_transaction
    ADD CONSTRAINT acg_transaction_pk PRIMARY KEY ( id,
                                                    account_no,
                                                    account_no1 );

CREATE TABLE acg_university (
```

```
    uni_id          NUMBER(4) NOT NULL,
    university_name VARCHAR2(30) NOT NULL
);

ALTER TABLE acg_university ADD CONSTRAINT acg_university_pk PRIMARY KEY ( uni_id );

CREATE TABLE acg_user (
    id           NUMBER NOT NULL,
    user_name    VARCHAR2(30) NOT NULL,
    password     VARCHAR2(30) NOT NULL,
    email        VARCHAR2(30) NOT NULL,
    is_superuser NUMBER NOT NULL,
    is_active    NUMBER NOT NULL
);

COMMENT ON COLUMN acg_user.user_name IS
    'User name for the user';

COMMENT ON COLUMN acg_user.password IS
    'password for the user';

COMMENT ON COLUMN acg_user.email IS
    'email id for the user';
```

```
COMMENT ON COLUMN acg_user.is_superuser IS
    'to indicate if the user is super user or not';

COMMENT ON COLUMN acg_user.is_active IS
    'to indicate if the account is active or not';

ALTER TABLE acg_user ADD CONSTRAINT acg_user_pk PRIMARY KEY ( id );

ALTER TABLE acg_account
    ADD CONSTRAINT fk_account_customer FOREIGN KEY ( cust_id )
        REFERENCES acg_customer ( cust_id );

ALTER TABLE acg_checkings
    ADD CONSTRAINT fk_checkings_account FOREIGN KEY ( account_id,
                                                      account_type )
        REFERENCES acg_account ( account_id,
                                 account_type );

ALTER TABLE acg_customer
    ADD CONSTRAINT fk_customer_user FOREIGN KEY ( id )
        REFERENCES acg_user ( id );
```

```
ALTER TABLE acg_home_insurance
   ADD CONSTRAINT fk_home_ins_insurance FOREIGN KEY ( company_id )
      REFERENCES acg_insurance ( company_id );

ALTER TABLE acg_home_loan
   ADD CONSTRAINT fk_home_loan_insurance FOREIGN KEY ( home_ins_acc_no )
      REFERENCES acg_home_insurance ( home_ins_acc_no );

ALTER TABLE acg_home_loan
   ADD CONSTRAINT fk_home_loan_loan FOREIGN KEY ( account_id,
                                                  account_type )
      REFERENCES acg_loan ( account_id,
                            account_type );

ALTER TABLE acg_loan
   ADD CONSTRAINT fk_loan_account FOREIGN KEY ( account_id,
                                                account_type )
      REFERENCES acg_account ( account_id,
                               account_type );

ALTER TABLE acg_personal_loan
   ADD CONSTRAINT fk_personal_loan_loan FOREIGN KEY ( account_id,
                                                      account_type )
```

```
        REFERENCES acg_loan ( account_id,
                              account_type );

ALTER TABLE acg_savings
    ADD CONSTRAINT fk_savings_account FOREIGN KEY ( account_id,
                                                    account_type )
        REFERENCES acg_account ( account_id,
                                 account_type );

ALTER TABLE acg_student_loan
    ADD CONSTRAINT fk_student_loan_loan FOREIGN KEY ( account_id,
                                                      account_type )
        REFERENCES acg_loan ( account_id,
                              account_type );

ALTER TABLE acg_student_uni
    ADD CONSTRAINT fk_student_uni_loan FOREIGN KEY ( stu_loan_id )
        REFERENCES acg_student_loan ( stu_loan_id );

ALTER TABLE acg_student_uni
    ADD CONSTRAINT fk_student_uni_university FOREIGN KEY ( uni_id )
        REFERENCES acg_university ( uni_id );
```

```
ALTER TABLE acg_transaction
   ADD CONSTRAINT fk_transaction_checkings FOREIGN KEY ( account_no1 )
      REFERENCES acg_checkings ( account_no );

ALTER TABLE acg_transaction
   ADD CONSTRAINT fk_transaction_savings FOREIGN KEY ( account_no )
      REFERENCES acg_savings ( account_no );

CREATE OR REPLACE TRIGGER arc_fkarc_5_acg_personal_loan BEFORE
   INSERT OR UPDATE OF account_id, account_type ON acg_personal_loan
   FOR EACH ROW
DECLARE
   d VARCHAR2(2);
BEGIN
   SELECT
      a.loan_type
   INTO d
   FROM
      acg_loan a
   WHERE
         a.account_id = :new.account_id
      AND a.account_type = :new.account_type;
```

```
   IF ( d IS NULL OR d <> 'PL' ) THEN

                raise_application_error(-20223, 'FK FK_PERSONAL_LOAN_LOAN in Table ACG_PERSONAL_LOAN violates Arc constraint on Table ACG_LOAN - discriminator column LOAN_TYPE doesn"t have value "PL"'

      );

   END IF;


EXCEPTION

   WHEN no_data_found THEN

      NULL;

   WHEN OTHERS THEN

      RAISE;

END;

/


CREATE OR REPLACE TRIGGER arc_fkarc_5_acg_home_loan BEFORE

   INSERT OR UPDATE OF account_id, account_type ON acg_home_loan

   FOR EACH ROW

DECLARE

   d VARCHAR2(2);

BEGIN

   SELECT

      a.loan_type

   INTO d
```

```
   FROM
      acg_loan a
   WHERE
         a.account_id = :new.account_id
      AND a.account_type = :new.account_type;

   IF ( d IS NULL OR d <> 'HL' ) THEN
                  raise_application_error(-20223, 'FK FK_HOME_LOAN_LOAN in Table
ACG_HOME_LOAN violates Arc constraint on Table ACG_LOAN - discriminator column
LOAN_TYPE doesn"t have value "HL"'
      );
   END IF;

EXCEPTION
   WHEN no_data_found THEN
      NULL;
   WHEN OTHERS THEN
      RAISE;
END;
/

CREATE OR REPLACE TRIGGER arc_fkarc_5_acg_student_loan BEFORE
   INSERT OR UPDATE OF account_id, account_type ON acg_student_loan
   FOR EACH ROW
```

```
DECLARE
   d VARCHAR2(2);
BEGIN
   SELECT
      a.loan_type
   INTO d
   FROM
      acg_loan a
   WHERE
         a.account_id = :new.account_id
      AND a.account_type = :new.account_type;


   IF ( d IS NULL OR d <> 'SL' ) THEN
                  raise_application_error(-20223, 'FK FK_STUDENT_LOAN_LOAN in Table
ACG_STUDENT_LOAN violates Arc constraint on Table ACG_LOAN - discriminator column
LOAN_TYPE doesn"t have value "SL"'
      );
   END IF;


EXCEPTION
   WHEN no_data_found THEN
      NULL;
   WHEN OTHERS THEN
      RAISE;
```

```
END;
/


CREATE OR REPLACE TRIGGER arc_fkarc_6_acg_loan BEFORE
   INSERT OR UPDATE OF account_id, account_type ON acg_loan
   FOR EACH ROW
DECLARE
   d VARCHAR2(2);
BEGIN
   SELECT
      a.account_type
   INTO d
   FROM
      acg_account a
   WHERE
         a.account_id = :new.account_id
      AND a.account_type = :new.account_type;


   IF ( d IS NULL OR d <> 'L' ) THEN
            raise_application_error(-20223, 'FK FK_LOAN_ACCOUNT in Table ACG_LOAN violates Arc constraint on Table ACG_ACCOUNT - discriminator column ACCOUNT_TYPE doesn"t have value "L"'
      );
   END IF;
```

```
EXCEPTION
   WHEN no_data_found THEN
      NULL;
   WHEN OTHERS THEN
      RAISE;
END;
/


CREATE OR REPLACE TRIGGER arc_fkarc_6_acg_savings BEFORE
   INSERT OR UPDATE OF account_id, account_type ON acg_savings
   FOR EACH ROW
DECLARE
   d VARCHAR2(2);
BEGIN
   SELECT
      a.account_type
   INTO d
   FROM
      acg_account a
   WHERE
         a.account_id = :new.account_id
      AND a.account_type = :new.account_type;
```

```
   IF ( d IS NULL OR d <> 'S' ) THEN

      raise_application_error(-20223, 'FK FK_SAVINGS_ACCOUNT in Table ACG_SAVINGS violates Arc constraint on Table ACG_ACCOUNT - discriminator column ACCOUNT_TYPE doesn"t have value "S"'

      );

   END IF;


EXCEPTION

   WHEN no_data_found THEN

      NULL;

   WHEN OTHERS THEN

      RAISE;

END;

/


CREATE OR REPLACE TRIGGER arc_fkarc_6_acg_checkings BEFORE

   INSERT OR UPDATE OF account_id, account_type ON acg_checkings

   FOR EACH ROW

DECLARE

   d VARCHAR2(2);

BEGIN

   SELECT

      a.account_type
```

```
   INTO d
   FROM
      acg_account a
   WHERE
         a.account_id = :new.account_id
      AND a.account_type = :new.account_type;

   IF ( d IS NULL OR d <> 'C' ) THEN
                  raise_application_error(-20223, 'FK FK_CHECKINGS_ACCOUNT in Table
ACG_CHECKINGS violates Arc constraint on Table ACG_ACCOUNT - discriminator column
ACCOUNT_TYPE doesn"t have value "C"'
      );
   END IF;

EXCEPTION
   WHEN no_data_found THEN
      NULL;
   WHEN OTHERS THEN
      RAISE;
END;
/
```

```
-- Oracle SQL Developer Data Modeler Summary Report:
--
-- CREATE TABLE                 14
-- CREATE INDEX                  1
-- ALTER TABLE                  36
-- CREATE VIEW                   0
-- ALTER VIEW                    0
-- CREATE PACKAGE                0
-- CREATE PACKAGE BODY           0
-- CREATE PROCEDURE              0
-- CREATE FUNCTION               0
-- CREATE TRIGGER                6
-- ALTER TRIGGER                 0
-- CREATE COLLECTION TYPE        0
-- CREATE STRUCTURED TYPE        0
-- CREATE STRUCTURED TYPE BODY   0
-- CREATE CLUSTER                0
-- CREATE CONTEXT                0
-- CREATE DATABASE               0
-- CREATE DIMENSION              0
-- CREATE DIRECTORY              0
-- CREATE DISK GROUP             0
-- CREATE ROLE                   0
```

-- CREATE ROLLBACK SEGMENT 0

-- CREATE SEQUENCE 0

-- CREATE MATERIALIZED VIEW 0

-- CREATE MATERIALIZED VIEW LOG 0

-- CREATE SYNONYM 0

-- CREATE TABLESPACE 0

-- CREATE USER 0

--

-- DROP TABLESPACE 0

-- DROP DATABASE 0

--

-- REDACTION POLICY 0

--

-- ORDS DROP SCHEMA 0

-- ORDS ENABLE SCHEMA 0

-- ORDS ENABLE OBJECT 0

--

-- ERRORS 0

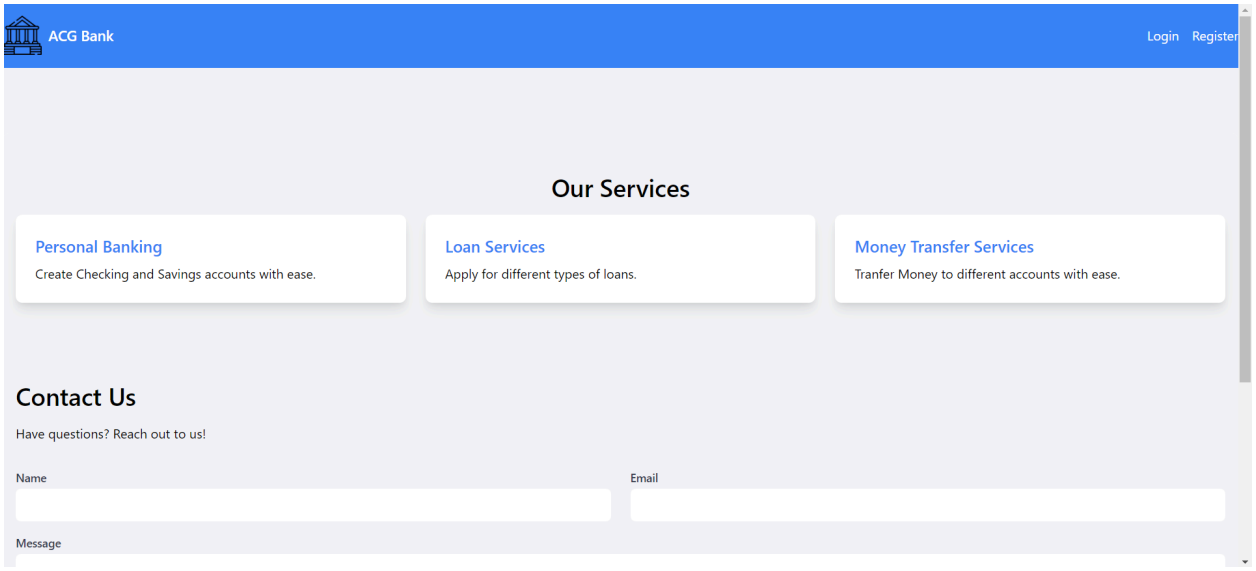
-- WARNINGS 0

List of tables, and total number of records of each table

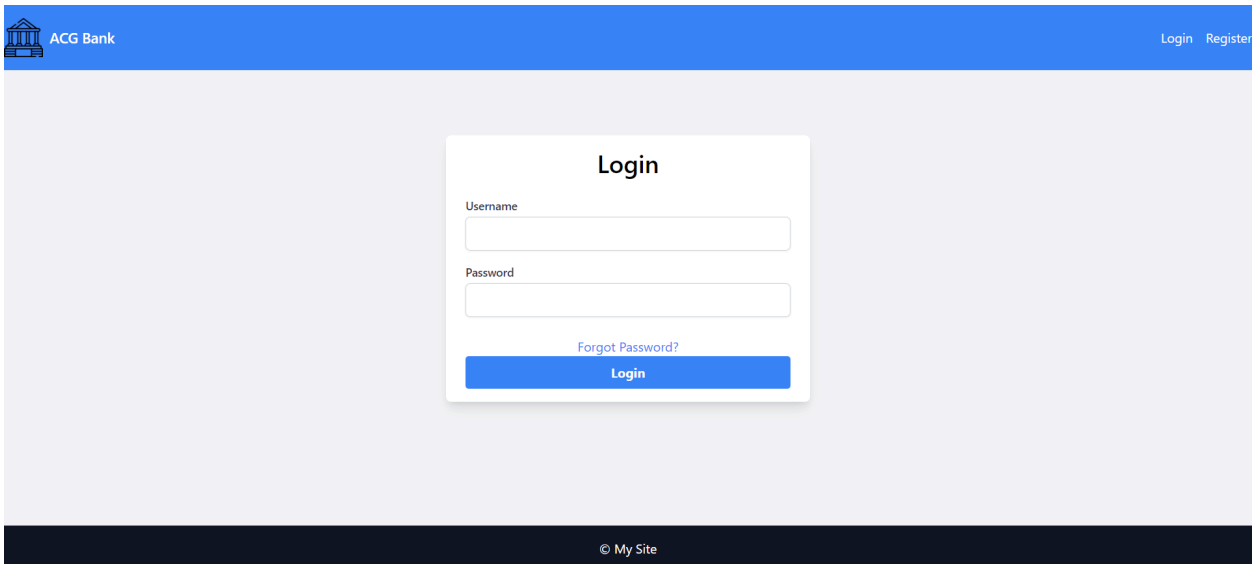
```
Model: Customer, Number of Rows: 16
Model: CustomUser, Number of Rows: 18
Model: Account, Number of Rows: 20
Model: SavingAccount, Number of Rows: 6
Model: CheckingAccount, Number of Rows: 8
Model: LoanAccount, Number of Rows: 6
Model: PersonalLoan, Number of Rows: 6
Model: StudentLoan, Number of Rows: 5
Model: University, Number of Rows: 8
Model: StudentUniversity, Number of Rows: 6
Model: Insurance, Number of Rows: 7
Model: HomeLoan, Number of Rows: 8
Model: HomeInsurance, Number of Rows: 9
Model: Transaction, Number of Rows: 16
>>>
```

Screenshots of some sessions, pages, menus of your Web Application

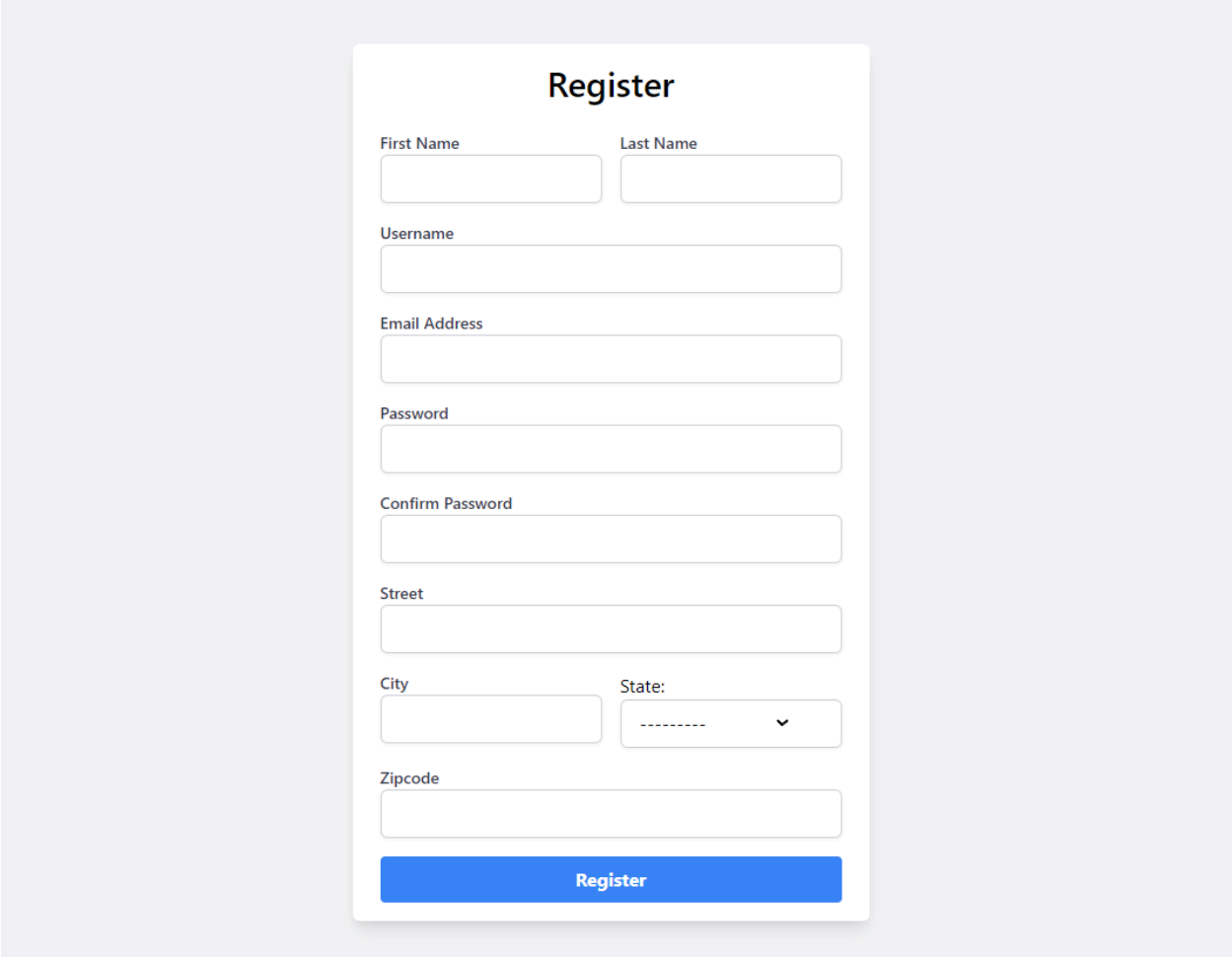
Landing page for the website



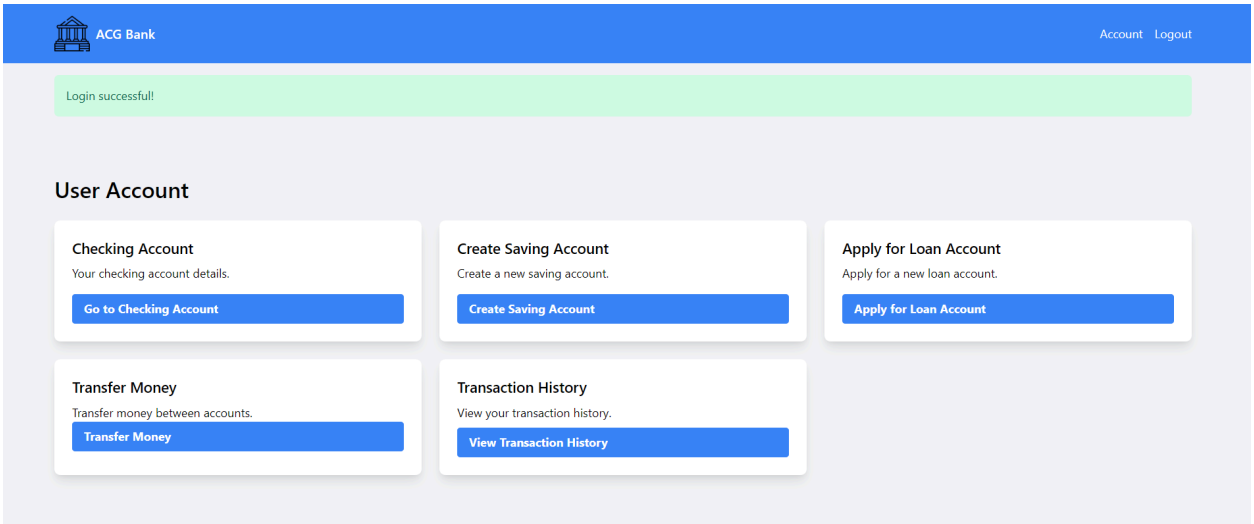
Login page



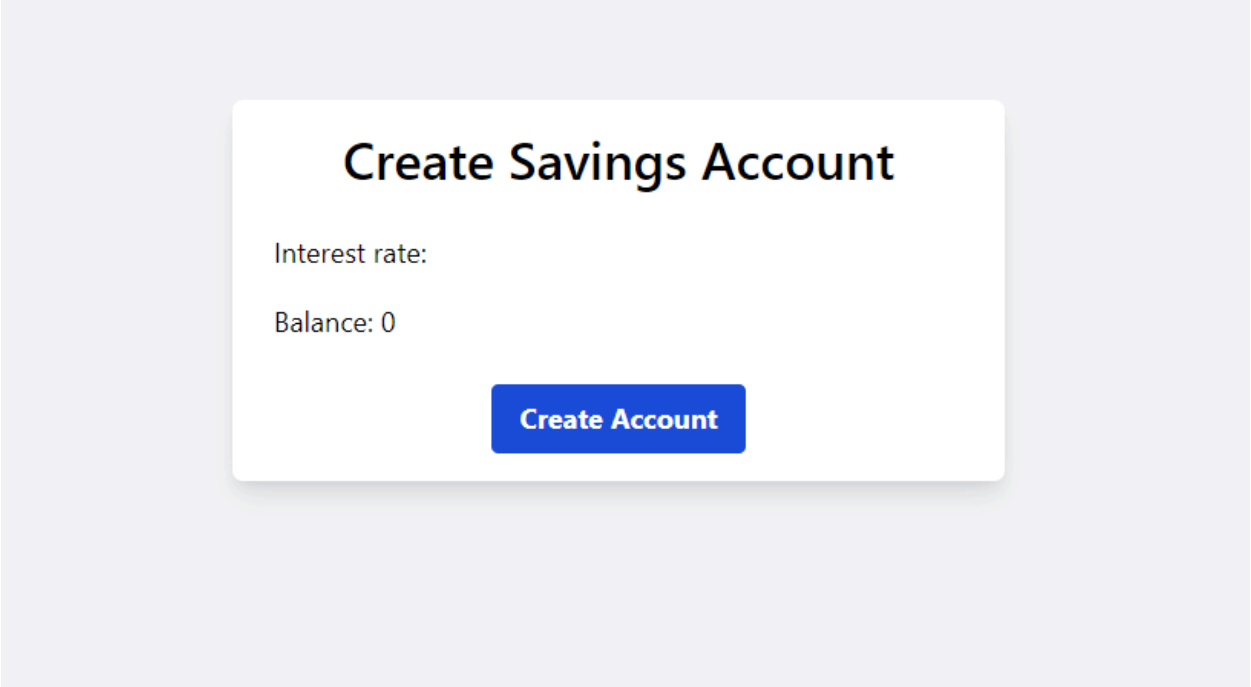
User registration page



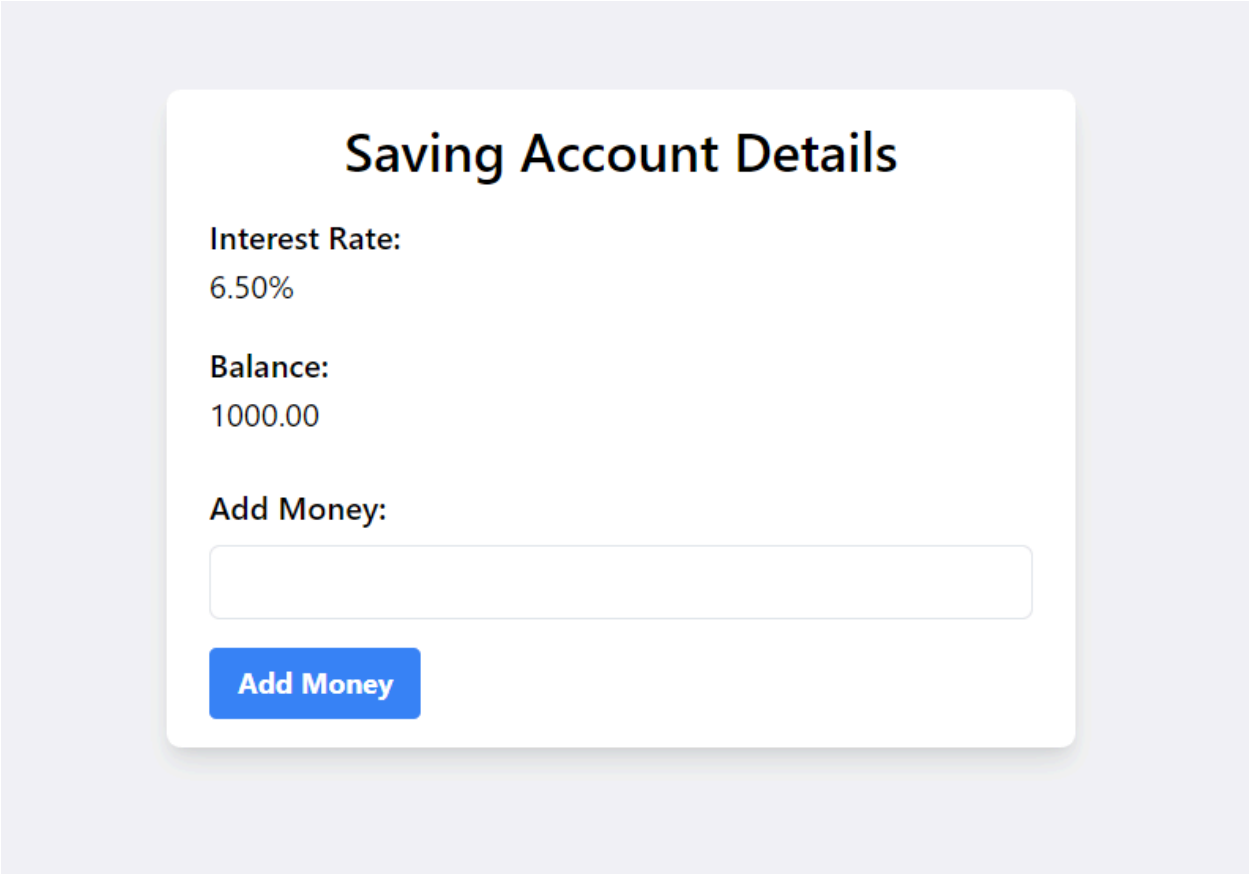
Account overview page



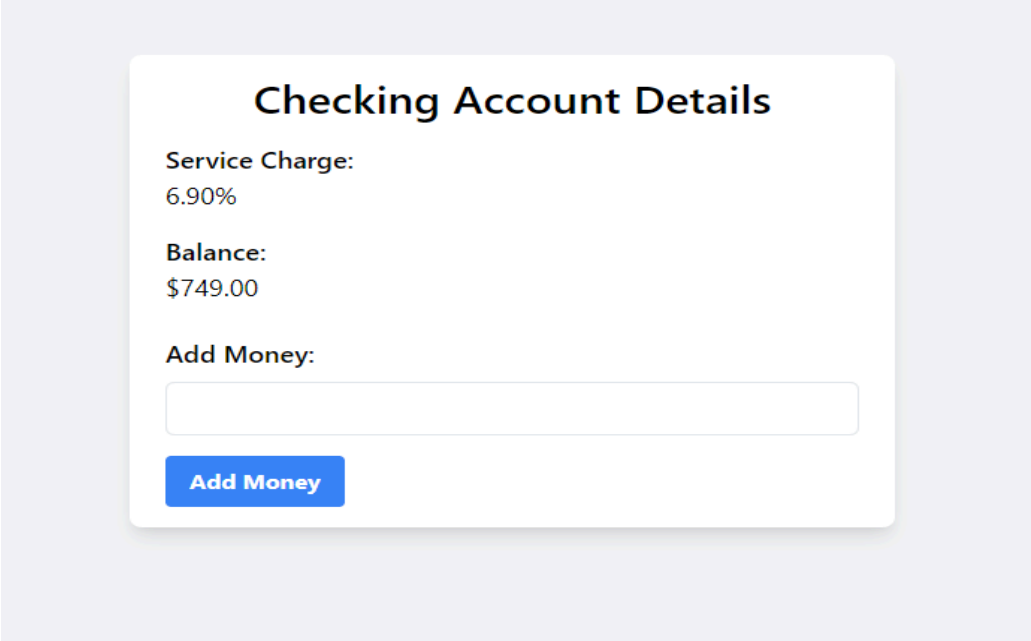
Create savings account page



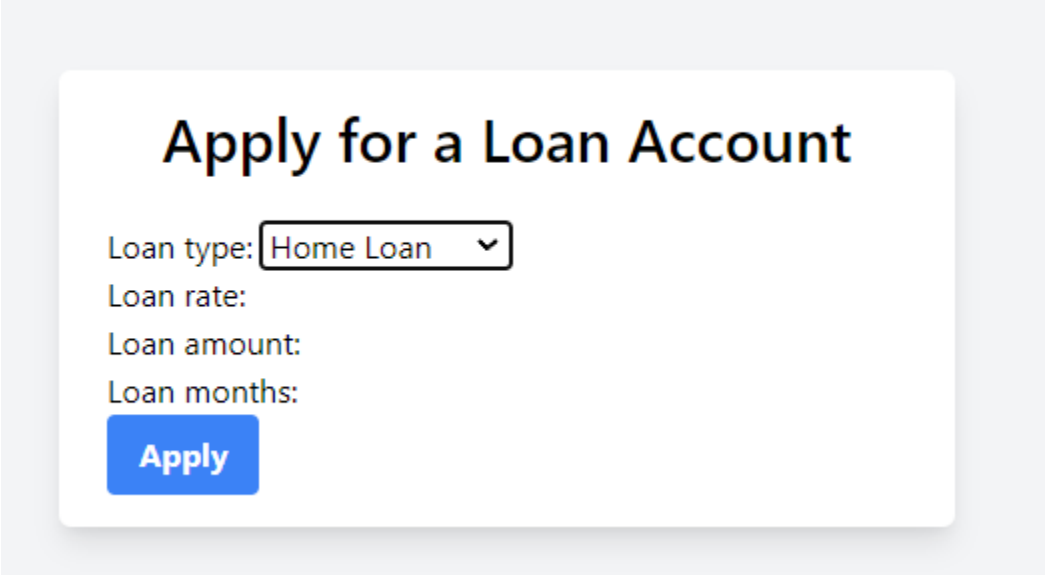
Savings Account details page



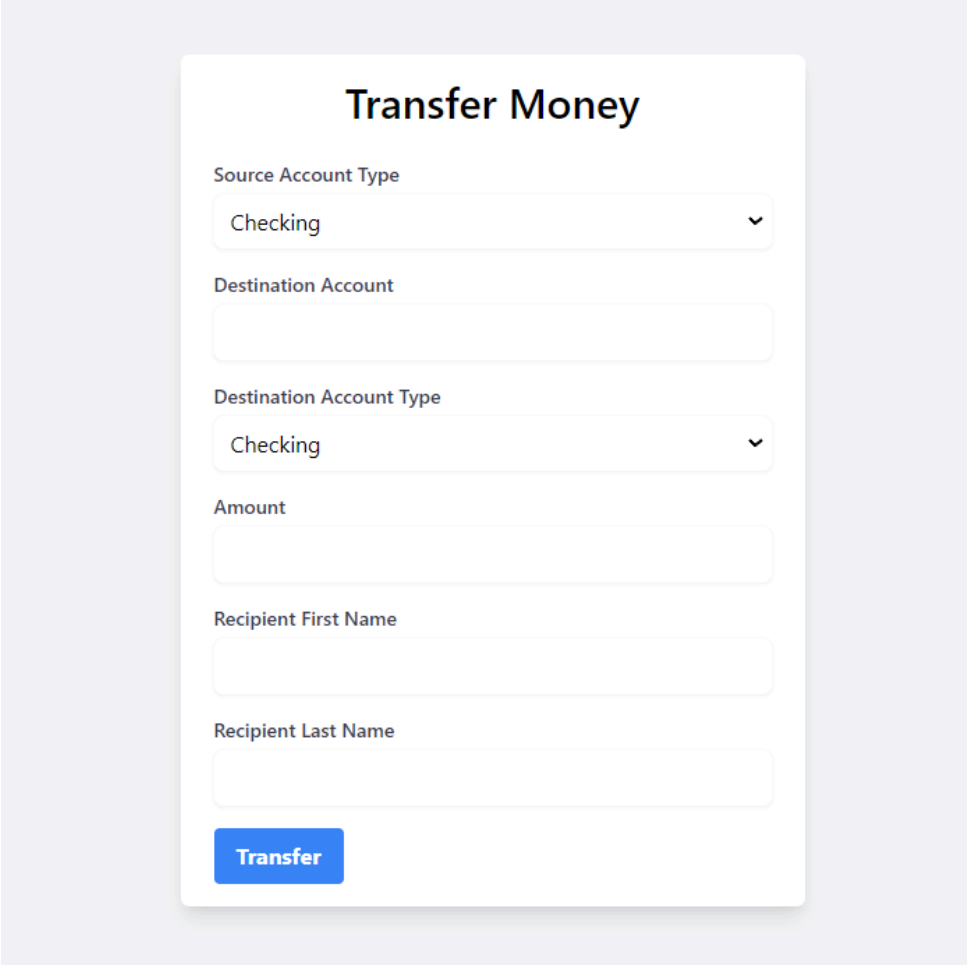
Checking account details page



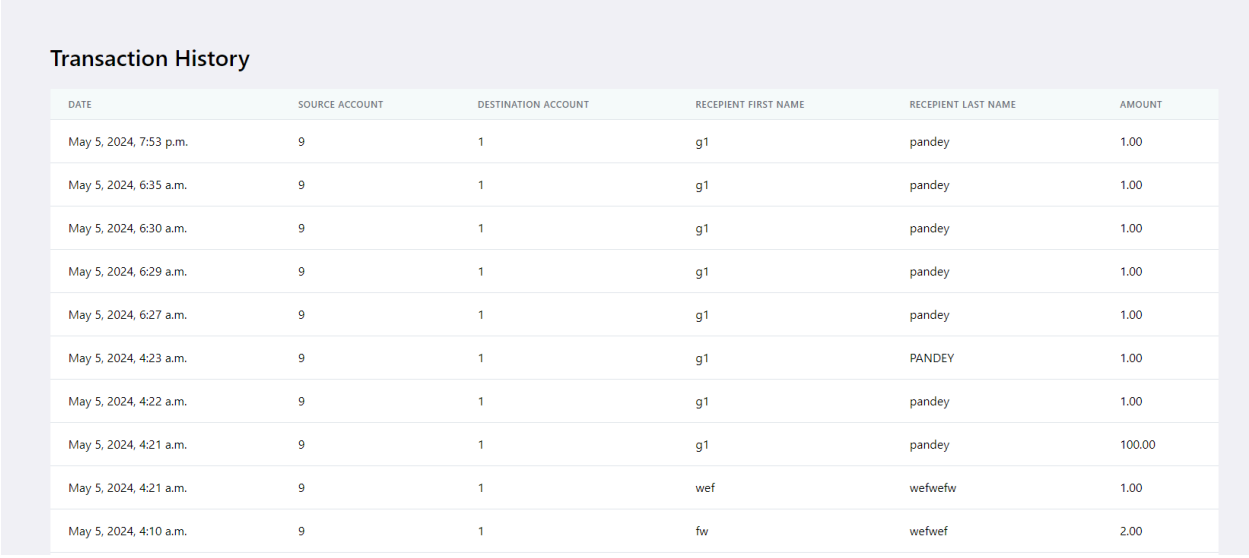
Apply for loan account



Money transfer page



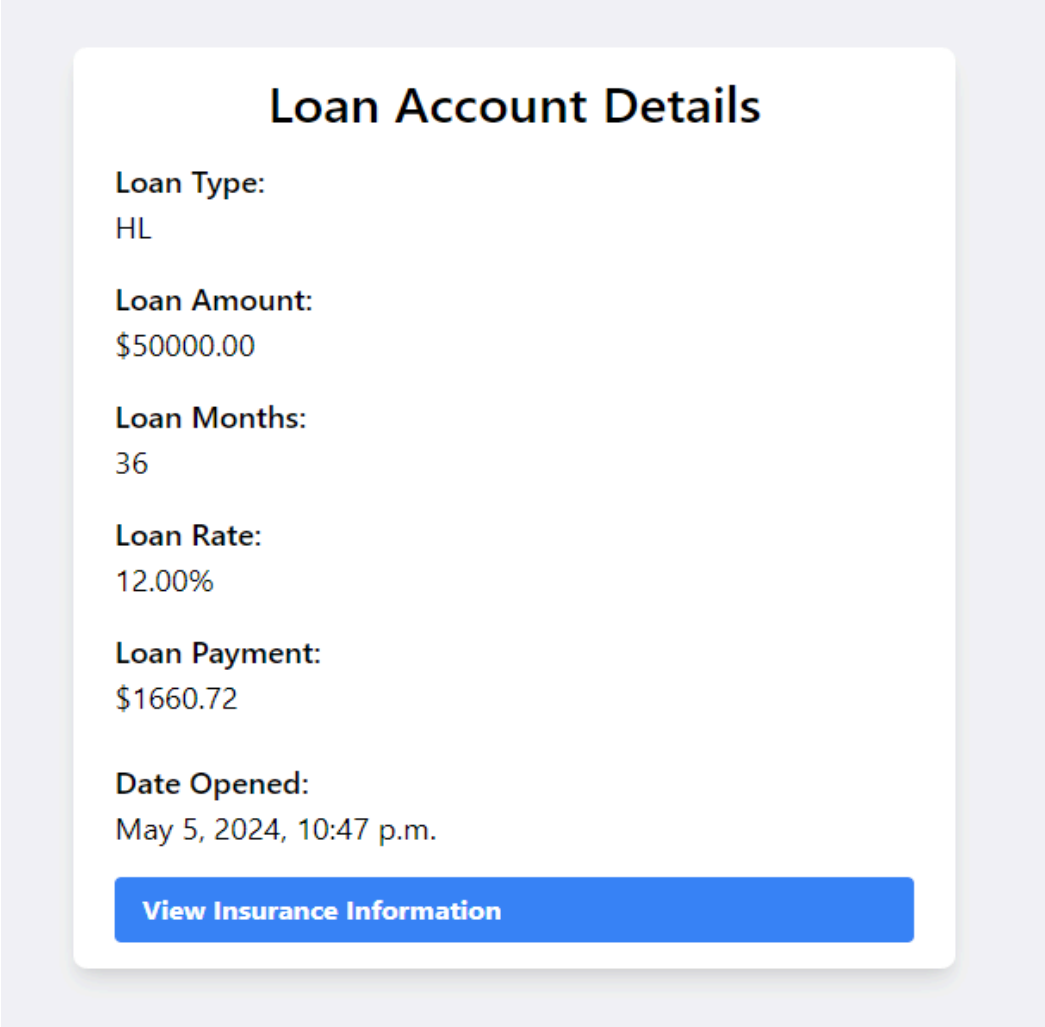
Transaction history page



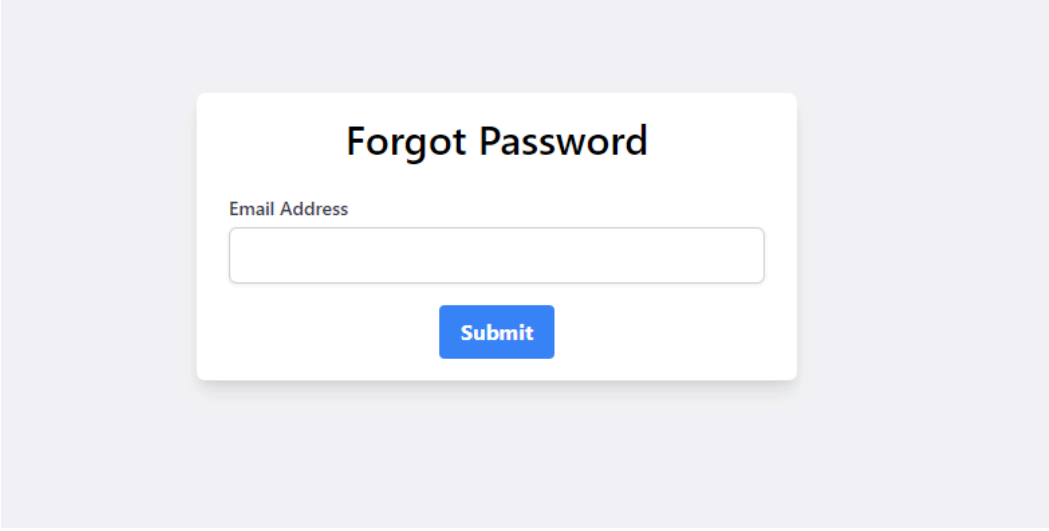
Transaction History

DATE	SOURCE ACCOUNT	DESTINATION ACCOUNT	RECEPIENT FIRST NAME	RECEPIENT LAST NAME	AMOUNT
May 5, 2024, 7:53 p.m.	9	1	g1	pandey	1.00
May 5, 2024, 6:35 a.m.	9	1	g1	pandey	1.00
May 5, 2024, 6:30 a.m.	9	1	g1	pandey	1.00
May 5, 2024, 6:29 a.m.	9	1	g1	pandey	1.00
May 5, 2024, 6:27 a.m.	9	1	g1	pandey	1.00
May 5, 2024, 4:23 a.m.	9	1	g1	PANDEY	1.00
May 5, 2024, 4:22 a.m.	9	1	g1	pandey	1.00
May 5, 2024, 4:21 a.m.	9	1	g1	pandey	100.00
May 5, 2024, 4:21 a.m.	9	1	wef	wefwefw	1.00
May 5, 2024, 4:10 a.m.	9	1	fw	wefwef	2.00

Loan Account details page



Forgot password page



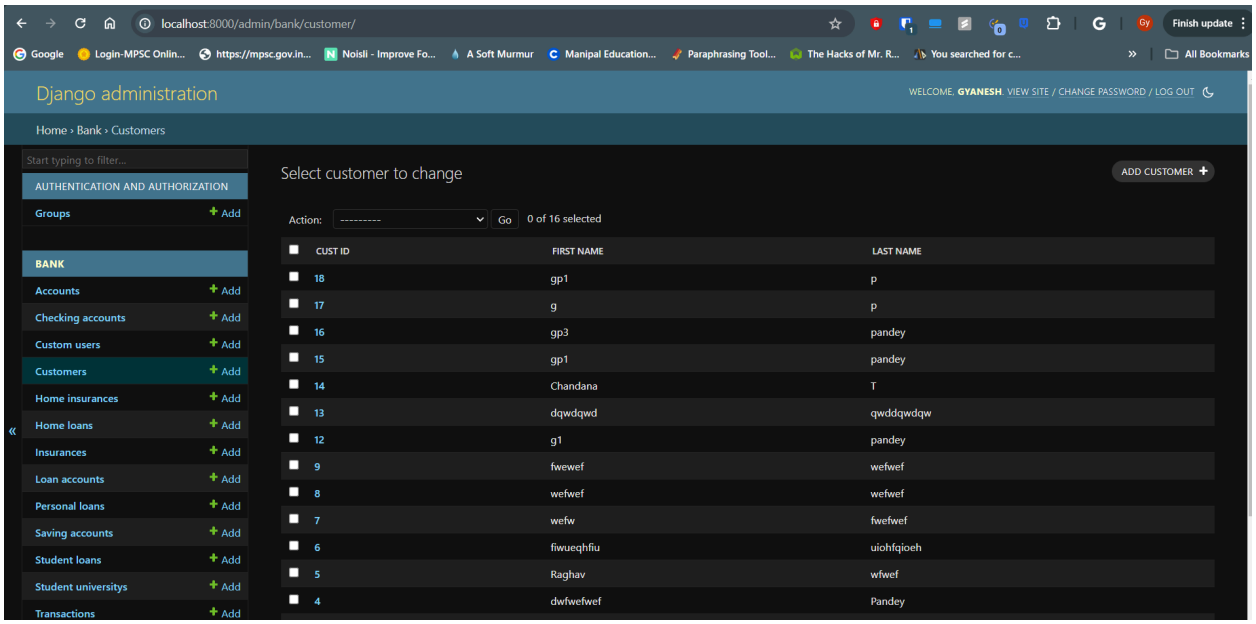
Details of security features that we have implemented on Web Application development.

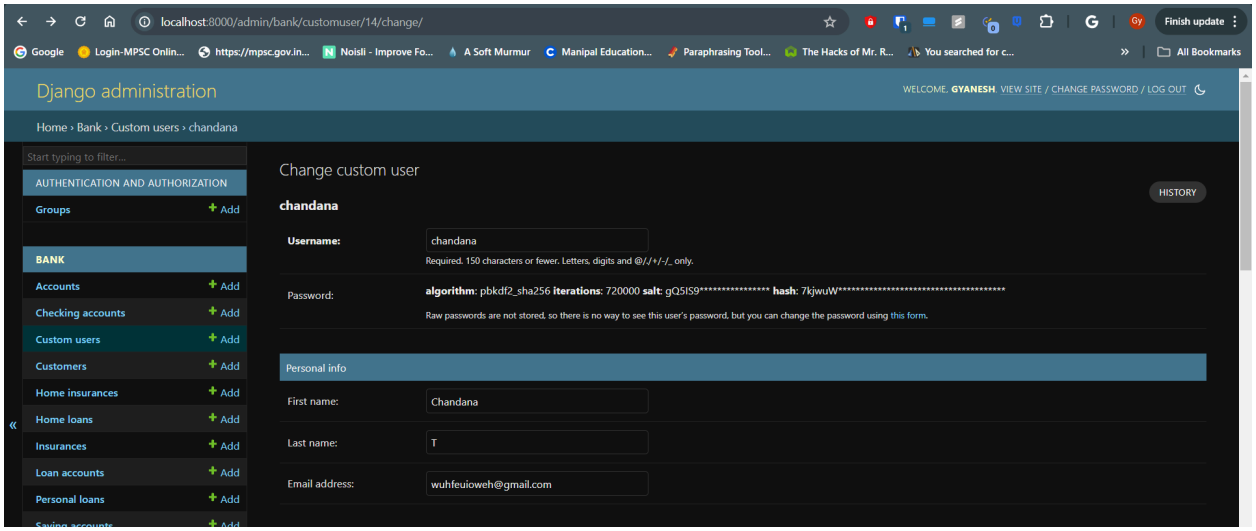
In our web application development, we have implemented robust security measures to enhance user authentication and data protection. One notable feature is the integration of One-Time Password (OTP) functionality for critical actions such as password resetting and money transfers. This additional layer of security adds an extra barrier against unauthorized access by requiring users to authenticate themselves through a unique OTP generated for each transaction. Furthermore, to mitigate the risk of OTP misuse, we have implemented a time-based expiration mechanism. Each OTP generated is stored in the cache for a limited duration of 3 minutes before being automatically invalidated.
In addition to OTP authentication, we have incorporated comprehensive validation features to ensure the integrity of user data. Our validation processes include thorough checks on registered customer information, enforcing criteria such as a minimum password length of 8 characters. By imposing strict validation rules, we aim to uphold data quality standards and prevent common security vulnerabilities such as weak passwords. These measures collectively contribute to safeguarding sensitive user information and fortifying our application against potential security threats.

Some of the security features implemented using Django's template

- **Cross-Site Request Forgery (CSRF) Protection:** We have integrated Django's built-in CSRF protection, which utilizes CSRF tokens to validate incoming requests, thereby preventing unauthorized form submissions from malicious websites.
- **Cross-Site Scripting (XSS) Protection:** Our application leverages Django's template system, which automatically escapes variables to prevent XSS attacks by converting potentially dangerous characters into their HTML entity equivalents.
- **SQL Injection Protection:** Utilizing Django's ORM system, we ensure protection against SQL injection attacks by automatically escaping parameters passed to database queries, thus enhancing the security of our database interactions.
- **Clickjacking Protection:** We have integrated Django's middleware to set the X-Frame-Options header, mitigating clickjacking attacks by preventing the application from being embedded in an iframe on another domain.
- **Password Hashing:** Our authentication system utilizes robust password hashing algorithms such as PBKDF2, bcrypt, or Argon2 to securely store user passwords in the database. Additionally, we leverage Django's utilities for password validation and resetting to enhance overall security.
- **User Authentication and Authorization:** We have implemented Django's authentication system to handle user authentication securely, including features such as session management, user permissions, and user groups, ensuring proper user access control.

- **Secure Cookies:** Our application's cookie settings are configured to ensure they are only transmitted over HTTPS connections, enhancing their security against interception by unauthorized parties.
- **Request Validation:** We utilize Django's built-in form and model validation mechanisms to validate user input and ensure data integrity, thereby mitigating various types of attacks, including injection attacks and data tampering.



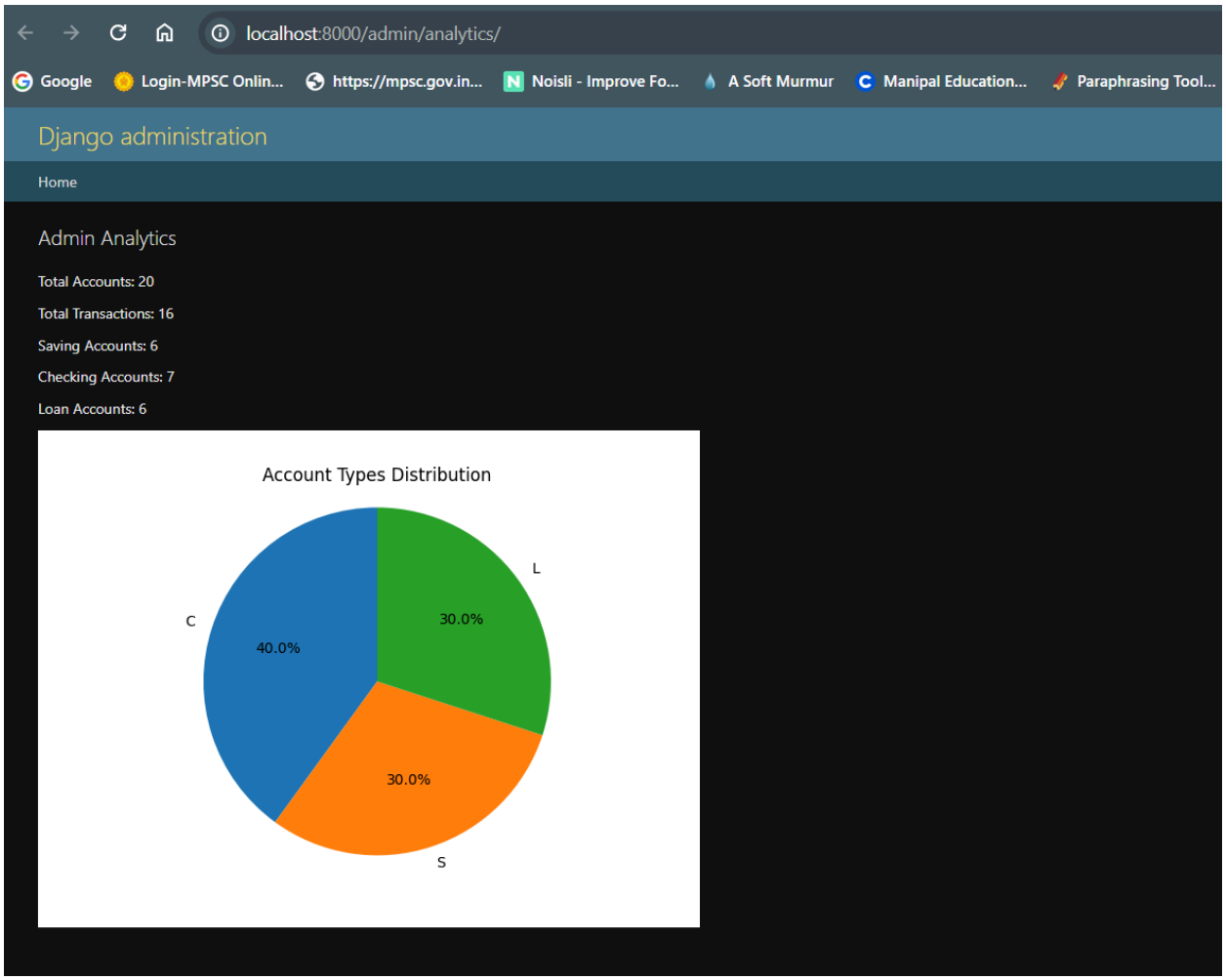


In the admin panel screenshots shown above, we demonstrate the robust security measures implemented within our system. Passwords are securely stored using salted and hashed encryption, ensuring the protection of user credentials.

Some of the additional features we implemented to enhance the functionality and security of the web application include:

- **Indexing Customer ID and Account Type:** We optimized database performance by indexing Customer ID and Account Type, as they are frequently accessed entities. This indexing strategy improved query execution times, resulting in faster data retrieval and improved overall system responsiveness.
- **Admin Analytics:** For admin analytics, a pie chart visualizes key metrics exclusively accessible to admin users, including the total number of accounts, overall transaction volume, and the distribution of different account types created. This graphical representation offers a comprehensive overview, allowing admins to swiftly grasp vital insights at a glance, facilitating informed decision-making and strategic planning.

  The below image shows the data visualization for admin analytics.



- **OTP Caching for Authentication:** To enhance security and user experience, each One-Time Password (OTP) generated during authentication is temporarily stored in the cache for a limited duration of 3 minutes. After this period, the OTP is automatically invalidated, mitigating the risk of unauthorized access and ensuring robust authentication mechanisms.
- **Security Checks for Password Reset:** To fortify the password reset process against unauthorized access, we implemented additional security checks, including the requirement of an OTP for password reset requests.

These features collectively contribute to the robustness, security, and scalability of the web application.

Lessons and Reflections from Project

Reflecting on the project, several lessons emerged that contributed to our growth and development as a team. Firstly, clear communication and effective collaboration were paramount. We learned the importance of establishing open channels for sharing ideas, addressing concerns, and aligning our efforts towards common goals. Throughout this project, debugging played a crucial role in refining our code and ensuring its functionality. Collaborating among the team members helped us overcome challenges and fostered a sense of unity in achieving project milestones.

Additionally, delving into the intricacies of design highlighted the significance of creating intuitive and user-friendly interfaces. We realized the importance of proper planning and designing before starting with development, as it lays a solid foundation for the entire project lifecycle. Leveraging technologies like Django, Python, HTML, CSS (with Tailwind CSS), and JavaScript enriched our technical skill set and empowered us to tackle complex development tasks with confidence. Visual Studio Code served as an invaluable tool in enhancing our productivity and code management throughout the project.

Moreover, the course led by the Professor provided an amazing foundation, tying everything we learned in class to the practical application of building a web application. This hands-on experience reinforced key concepts and demonstrated the real-world relevance of our coursework. We are grateful for the guidance and support provided by the Professor and TAs, which greatly contributed to the success of our project.

In summary, the project provided valuable insights into effective project management practices, technical skill development, and collaboration strategies. Leveraging lessons learned, such as the

importance of testing and debugging, we adopted a proactive approach that enabled successful project completion.

For future development, we envision incorporating more features for admin functionality, enhancing security measures through extensive logging of user activity for audit purposes, and implementing better data visualizations to provide deeper insights into the application's performance and user interactions. These enhancements will further elevate the functionality and user experience of our web application, ensuring its continued success and relevance in the dynamic digital landscape.

Business analysis with 6 SQLs using your project data.

Q1) Table joins with at least 3 tables in join.

```
SELECT c.first_name, c.last_name, a.account_type, l.loan_amount
FROM acg_customer c
JOIN acg_account a ON c.cust_id = a.cust_id
JOIN acg_loan l ON a.account_id = l.account_id AND a.account_type = l.account_type;
```

```
SELECT c.first_name, c.last_name, a.account_type, l.loan_amount
FROM acg_customer c
JOIN acg_account a ON c.cust_id = a.cust_id
JOIN acg_loan l ON a.account_id = l.account_id AND a.account_type = l.account_type;
```

100% 1:452

Result Grid Filter Rows: Search Export:

first_name	last_name	account_ty...	loan_amount
Jane	Smith	L	15000.00
Michael	Johnson	L	25000.00
Emily	Brown	L	75000.00
David	Jones	L	45000.00
Sarah	Martinez	L	65000.00
Daniel	Garcia	L	95000.00
Olivia	Rodriguez	L	75000.00
James	Lopez	L	15000.00
Sophia	Perez	L	45000.00
Alexander	Williams	L	85000.00
Ava	Wilson	L	35000.00
Ethan	Taylor	L	550000.00
Mia	Jackson	L	105000.00
Isabella	White	L	76700.00
Michael	Harris	L	65000.00
Emma	Martin	L	7000.00
Noah	Thompson	L	75000.00
Liam	Garcia	L	5000.00
Charlotte	Lee	L	15000.00
Carol	Williams	L	10000.00
David	Jones	L	23000.00
Eva	Brown	L	11000.00
Frank	Davis	L	12000.00
Grace	Miller	L	15000.00
Henry	Wilson	L	31000.00
Ivy	Moore	L	1000.00
Jack	Taylor	L	6000.00

Result 14

This query retrieves the first name, last name, account type, and loan amount of customers who have loans, enabling the bank to understand the loan details of its customers.

Q2) Multi-row subquery

```
SELECT c.first_name, c.last_name, c.city, c.state,
        (SELECT COUNT(*) FROM acg_account a WHERE a.cust_id = c.cust_id) AS num_accounts
FROM acg_customer c;
```

```
SELECT c.first_name, c.last_name, c.city, c.state,
457         (SELECT COUNT(*) FROM acg_account a WHERE a.cust_id = c.cust_id) AS num_accounts
458     FROM acg_customer c;
459
```

100% 21:458

Result Grid Filter Rows: Search Export:

first_name	last_name	city	state	num_accounts
John	Doe	Anytown	NY	2
Jane	Smith	Othertown	CA	2
Michael	Johnson	Anycity	TX	2
Emily	Brown	Sometown	FL	1
David	Jones	Anothercity	IL	2
Sarah	Martinez	Yetanothercity	WA	2
Daniel	Garcia	Someplace	GA	2
Olivia	Rodriguez	Nowhere	MA	2
James	Lopez	Anyplace	AZ	2
Sophia	Perez	Everywhere	NV	2
Alexander	Williams	Nothere	CO	2
Ava	Wilson	Somecity	OR	2
Ethan	Taylor	Anotheplace	OH	3
Mia	Jackson	Anyplaceelse	UT	1
Isabella	White	Somewhere	KS	3
Michael	Harris	Thisplace	TN	2
Emma	Martin	Thatplace	WI	1
Noah	Thompson	Here	ID	2
Liam	Garcia	There	HI	2
Charlotte	Lee	Everywhere	AK	2
Alice	Smith	Los Angeles	CA	3
Bob	Johnson	Houston	TX	3
Carol	Williams	Miami	FL	3
David	Jones	Chicago	IL	3
Eva	Brown	Seattle	WA	1
Frank	Davis	Atlanta	GA	1
Grace	Miller	Boston	MA	1
Henry	Wilson	Phoenix	AZ	1
Ivy	Moore	Las Vegas	NV	1
Jack	Taylor	Denver	CO	2

Result 15

This query counts the number of accounts each customer has, helping the bank to understand the customer distribution based on the number of accounts they hold.

Q3) Correlated subquery.

SELECT c.first_name, c.last_name, c.city, c.state,
(SELECT MAX(l.loan_amount) FROM acg_loan l WHERE l.account_id = a.account_id AND l.account_type = a.account_type) AS max_loan_amount
FROM acg_customer c
JOIN acg_account a ON c.cust_id = a.cust_id;

```
461 •  SELECT c.first_name, c.last_name, c.city, c.state,
462           (SELECT MAX(l.loan_amount) FROM acg_loan l WHERE l.account_id = a.account_id AND l.account_ty
463    FROM acg_customer c
464    JOIN acg_account a ON c.cust_id = a.cust_id;
465
466
```

00% 1:465

Result Grid Filter Rows: Search Export:

first_name	last_name	city	state	max_loan_amount
John	Doe	Anytown	NY	NULL
John	Doe	Anytown	NY	NULL
Jane	Smith	Othertown	CA	NULL
Jane	Smith	Othertown	CA	15000.00
Michael	Johnson	Anycity	TX	25000.00
Michael	Johnson	Anycity	TX	NULL
Emily	Brown	Sometown	FL	75000.00
David	Jones	Anothercity	IL	NULL
David	Jones	Anothercity	IL	45000.00
Sarah	Martinez	Yetanothercity	WA	65000.00
Sarah	Martinez	Yetanothercity	WA	NULL
Daniel	Garcia	Someplace	GA	95000.00
Daniel	Garcia	Someplace	GA	NULL
Olivia	Rodriguez	Nowhere	MA	NULL
Olivia	Rodriguez	Nowhere	MA	75000.00
James	Lopez	Anyplace	AZ	15000.00
James	Lopez	Anyplace	AZ	NULL
Sophia	Perez	Everywhere	NV	NULL
Sophia	Perez	Everywhere	NV	45000.00
Alexander	Williams	Nothere	CO	85000.00
Alexander	Williams	Nothere	CO	NULL
Ava	Wilson	Somecity	OR	NULL
Ava	Wilson	Somecity	OR	35000.00
Ethan	Taylor	Anotheplace	OH	NULL
Ethan	Taylor	Anotheplace	OH	550000.00
Ethan	Taylor	Anotheplace	OH	NULL

Result 16

This query finds the maximum loan amount for each customer, assisting the bank in identifying customers with the highest loan amounts.

Q4) SET operator query.

```
SELECT account_id, account_type
FROM acg_savings
UNION
SELECT account_id, account_type
FROM acg_checkings;
```

```
495 •   SELECT account_id, account_type
496     FROM acg_savings
497     UNION
498     SELECT account_id, account_type
499     FROM acg_checkings;
500
```

100% 1:500

Result Grid Filter Rows: Search Export:

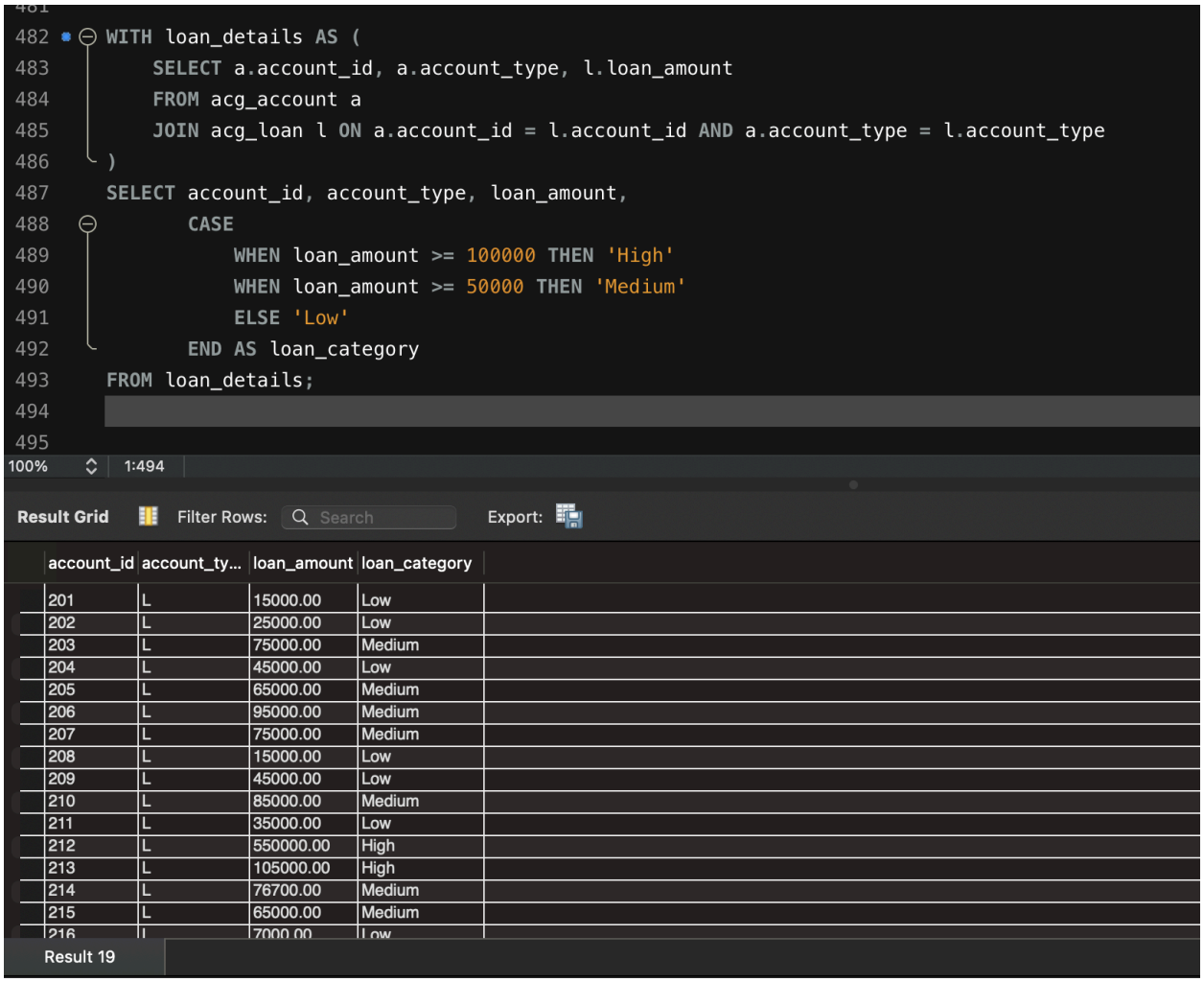
account_id	account_ty...
202	S
205	S
208	S
210	S
212	S
215	S
218	S
221	S
214	S
206	S
220	S
222	S
223	S
229	S
232	S
261	S
201	C
204	C
207	C
209	C
211	C
214	C
217	C
219	C
221	C
212	C

Result 20

This query combines the account IDs and types from both the savings and checkings accounts tables. It helps in identifying all the accounts across both types, which could be useful for various analysis or reporting purposes.

Q5) Query with in-line view or WITH clause

```
WITH loan_details AS (
   SELECT a.account_id, a.account_type, l.loan_amount
   FROM acg_account a
   JOIN acg_loan l ON a.account_id = l.account_id AND a.account_type = l.account_type
)
SELECT account_id, account_type, loan_amount,
      CASE
         WHEN loan_amount >= 100000 THEN 'High'
         WHEN loan_amount >= 50000 THEN 'Medium'
         ELSE 'Low'
      END AS loan_category
FROM loan_details;
```



```
WITH loan_details AS (
    SELECT a.account_id, a.account_type, l.loan_amount
    FROM acg_account a
    JOIN acg_loan l ON a.account_id = l.account_id AND a.account_type = l.account_type
)
SELECT account_id, account_type, loan_amount,
       CASE
           WHEN loan_amount >= 100000 THEN 'High'
           WHEN loan_amount >= 50000 THEN 'Medium'
           ELSE 'Low'
       END AS loan_category
FROM loan_details;
```

100% 1:494

Result Grid | Filter Rows: Search | Export:

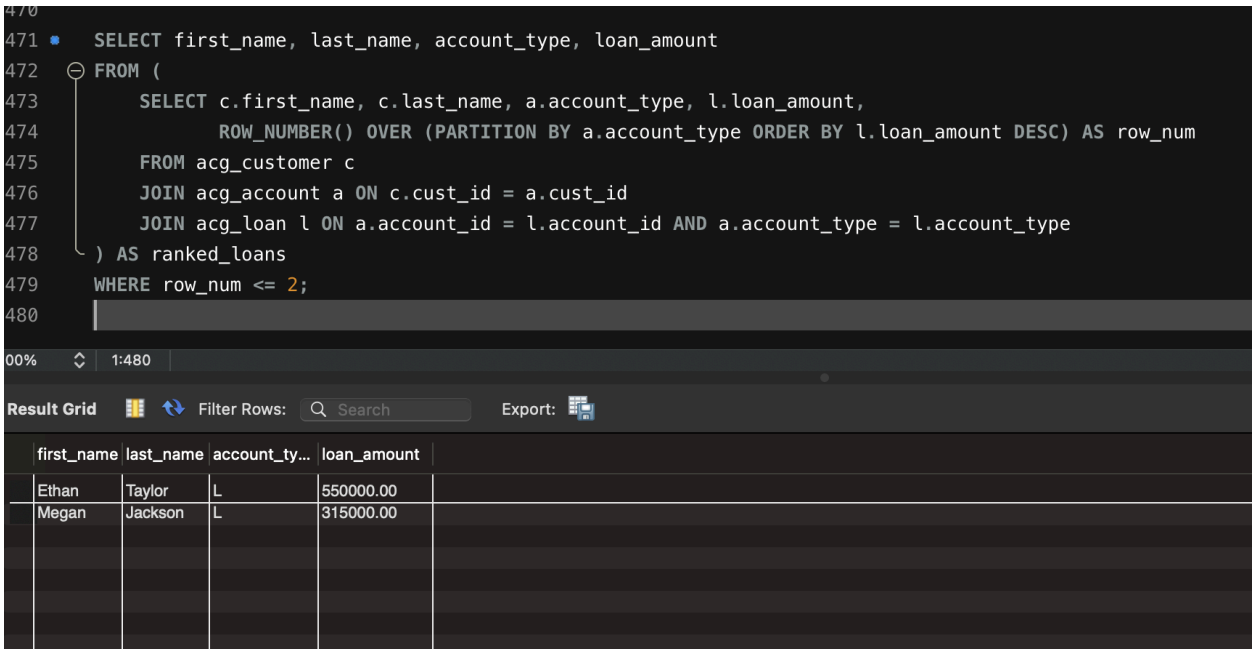
account_id	account_ty...	loan_amount	loan_category
201	L	15000.00	Low
202	L	25000.00	Low
203	L	75000.00	Medium
204	L	45000.00	Low
205	L	65000.00	Medium
206	L	95000.00	Medium
207	L	75000.00	Medium
208	L	15000.00	Low
209	L	45000.00	Low
210	L	85000.00	Medium
211	L	35000.00	Low
212	L	550000.00	High
213	L	105000.00	High
214	L	76700.00	Medium
215	L	65000.00	Medium
216	L	7000.00	Low

Result 19

This query categorizes loans into high, medium, or low categories based on their amounts, providing insights into the distribution of loan amounts among customers.

Q6) TOP-N/BOTTOM-N query

```
SELECT first_name, last_name, account_type, loan_amount
FROM (
   SELECT c.first_name, c.last_name, a.account_type, l.loan_amount,
          ROW_NUMBER() OVER (PARTITION BY a.account_type ORDER BY l.loan_amount
DESC) AS row_num
   FROM acg_customer c
   JOIN acg_account a ON c.cust_id = a.cust_id
   JOIN acg_loan l ON a.account_id = l.account_id AND a.account_type = l.account_type
) AS ranked_loans
WHERE row_num <= 2;
```



This query retrieves the top 2 loan amounts for each account type, assisting the bank in identifying customers with the highest loans in each category.