1.) Our personal finance management app serves as an ultimate companion to a user's financial literacy. The project would have features to store user's information such as:

- Name
- Contact information
- Existing bank accounts
- Existing credit/debit cards
- Expenses
- Income
- Financial goal

Our project will feature a relational database that will help manage and keep track of:

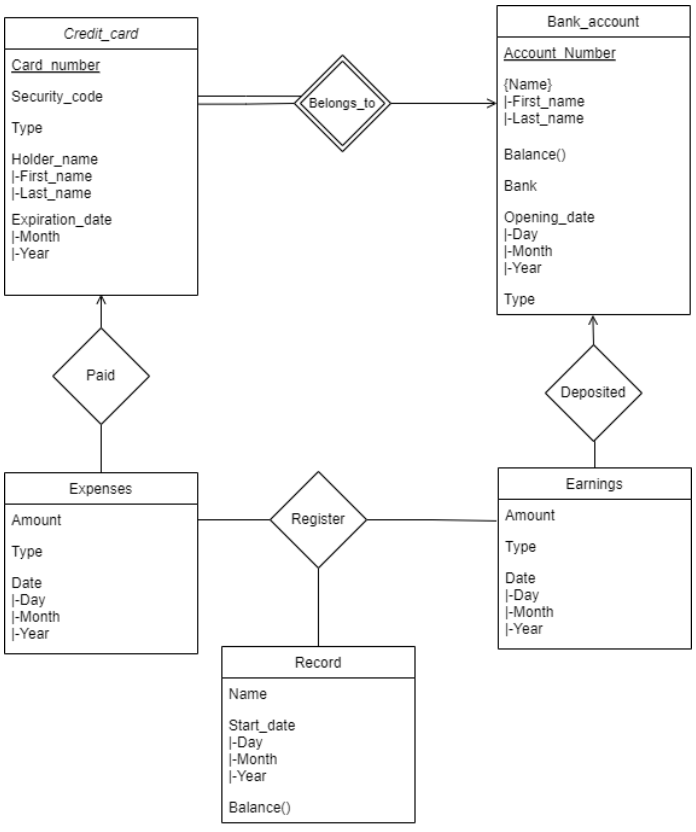
- User ID
- User's personal information
- User's accounts
- User's credit/debit cards
- Etc…

2.) User Requirements; High Level Operations

Some of the operations that our application will allow are:

- Add a payment made with your credit card
- Remove a payment that has been cancelled
- Change your credit card or bank information
- Read credit cards and balances

3.) E-R Diagram:



4.) Data Sources

Our app relies mostly on information provided directly by the user.

Users will manually enter their expenses, credit cards, bank accounts, and income into the app.

To showcase the full functionality of our system, we'll also include some preloaded data in the database.

This data will be carefully created using specialized tools to ensure it fits perfectly within our database's design and constraints.

5.) SQL Queries

DDL is used in these queries to define the structure of our database by creating the for bank_account, credit_card, movement, expense, record, earnings, and register.

```
-- create tables

create table bank_account (

        account_number int,

   first_name varchar(30) not null,

   last_name varchar(30) not null,

   bank varchar(30) not null,

   opening_date date not null,

   account_type varchar(20) not null,

   primary key (account_number)

);


create table credit_card (

        card_number int,
```

```
    security_code numeric(4) not null,
    card_type varchar(20) not null,
    first_name varchar(30),
    last_name varchar(30),
    expiration_date date not null,
    my_account int,
    primary key (card_number),
    foreign key (my_account) references bank_account(account_number) on delete set null
on update cascade
);

create table movement (
        movement_id int auto_increment,
    amount numeric(20,2) not null,
    movement_date date not null,
    my_account int,
    primary key (movement_id),
    foreign key (my_account) references bank_account(account_number) on delete set null
on update cascade
);

create table expense (
        expense_id int auto_increment,
    expense_type varchar(20) not null,
```

```
    my_card int,
    primary key (expense_id),
    foreign key (my_card) references credit_card(card_number) on delete set null on update cascade
);

create table earning (
        earning_id int auto_increment,
    earning_type varchar(20) not null,
    primary key (earning_id)
);

create table record (
        record_id int auto_increment,
        record_name varchar(30) not null unique,
    start_date date not null,
    primary key(record_id)
);

create table register (
        my_movement int,
    my_record int,
    foreign key (my_movement) references movement(movement_id) on delete cascade on update cascade,
```

foreign key (my_record) references record(record_id) on delete cascade on update cascade,

primary key (my_movement, my_record)

);

More of our Queries:

```
-- definition of functions
DELIMITER $$
CREATE FUNCTION get_account_balance(account_number INT)
RETURNS DECIMAL(20,2)
DETERMINISTIC
BEGIN
   DECLARE total_balance DECIMAL(20,2);
   SELECT COALESCE(SUM(amount), 0) INTO total_balance
   FROM movement
   WHERE my_account = account_number;
   RETURN total_balance;
END$$
DELIMITER ;

DELIMITER $$
CREATE FUNCTION get_record_balance(name_record VARCHAR(30))
RETURNS DECIMAL(20,2)
DETERMINISTIC
BEGIN
   DECLARE total_balance DECIMAL(20,2);
   SELECT COALESCE(SUM(amount), 0) INTO total_balance
   FROM movement
   WHERE (SELECT my_record FROM register WHERE my_movement = movement_id) =
(SELECT record_id FROM record WHERE record_name = name_record);
   RETURN total_balance;
END$$
DELIMITER ;
```

The **DML** queries are used to manipulate the data by inserting, updating, deleting, and retrieving data. Some examples of DML queries we used in our code are:

-- insert examples

INSERT INTO bank_account VALUES

(1, 'John', 'Doe', 'Chase Bank', '2024-10-21', 'Savings'),

(2, 'Jane', 'Smith', 'Bank of America', '2024-10-22', 'Savings'),

(3, 'Alice', 'Johnson', 'Wells Fargo', '2024-10-24', 'Checking'),

(4, 'Michael', 'Brown', 'Citibank', '2024-09-15', 'Savings'),

(5, 'Emma', 'Davis', 'Chase Bank', '2024-08-30', 'Savings'),

(6, 'Liam', 'Garcia', 'Bank of America', '2024-07-14', 'Checking'),

(7, 'Olivia', 'Martinez', 'Wells Fargo', '2024-06-18', 'Savings'),

(8, 'Noah', 'Lee', 'Citibank', '2024-05-25', 'Savings'),

(9, 'Sophia', 'Taylor', 'Chase Bank', '2024-04-10', 'Checking'),

(10, 'James', 'Anderson', 'Bank of America', '2024-03-05', 'Savings');

-- Inserting examples for credit_card with real names

INSERT INTO credit_card VALUES

(1, 1234, 'Credit', 'John', 'Doe', '2028-09-01', 1),

(2, 5678, 'Credit', 'Jane', 'Smith', '2029-10-01', 2),

(3, 9101, 'Debit', 'Alice', 'Johnson', '2027-12-01', 3),

(4, 1112, 'Credit', 'Michael', 'Brown', '2026-11-15', 4),

(5, 1314, 'Credit', 'Emma', 'Davis', '2027-10-18', 5),

(6, 1516, 'Debit', 'Liam', 'Garcia', '2026-09-21', 6),

(7, 1718, 'Credit', 'Olivia', 'Martinez', '2028-08-12', 7),

(8, 1920, 'Debit', 'Noah', 'Lee', '2027-07-07', 8),

(9, 2122, 'Credit', 'Sophia', 'Taylor', '2029-06-05', 9),

(10, 2324, 'Debit', 'James', 'Anderson', '2027-05-03', 10);

The rest of the queries:

```
SELECT get_account_balance(1);
insert into movement values
(1,21.21,'2024-10-04',1);
insert into earning values
(last_insert_id(),'Test');
insert into movement values
(2,-2,'2024-10-04',1);
insert into expense values
(last_insert_id(), 'Test', 1);
insert into record values
(1,'Test','2024-10-04');
insert into register values
(1,1);
insert into register values
(2,1);
SELECT get_record_balance('Test');

/* Deleting specific entries from bank_account and credit_card tables */

/* Deleting based on account_number */
DELETE FROM bank_account
WHERE account_number = 3;

DELETE FROM bank_account
WHERE account_number = 8;

/* Deleting based on card_number */
DELETE FROM credit_card
WHERE card_number = 2;

DELETE FROM credit_card
WHERE card_number = 7;

/* Deleting based on movement_id */
DELETE FROM movement
WHERE movement_id = 2;

/* Allowing updates/deletions without strict WHERE conditions */
SET SQL_SAFE_UPDATES = 0;
```

```
/* Deleting an entry from the bank_account table using the person's name */
DELETE FROM bank_account
WHERE first_name = 'Emma' AND last_name = 'Davis';

/* Deleting an entry from the credit_card table using the person's name */
DELETE FROM credit_card
WHERE first_name = 'John' AND last_name = 'Doe';

/* Verify remaining records by querying the tables */
SELECT * FROM bank_account;
SELECT * FROM credit_card;

-- join examples

SELECT * from bank_account inner join credit_card on credit_card.my_account =
bank_account.account_number; -- inner join bank credit

SELECT * from bank_account left join credit_card on credit_card.my_account =
bank_account.account_number; -- left join

SELECT * from bank_account right join credit_card on credit_card.my_account =
bank_account.account_number; -- right join

SELECT * from bank_account left join credit_card on credit_card.my_account =
bank_account.account_number
UNION
SELECT * from bank_account right join credit_card on credit_card.my_account =
bank_account.account_number; -- full join

SELECT * from register inner join movement on register.my_movement =
movement.movement_id; -- inner join register movement

SELECT * from register, movement, record where register.my_movement =
movement.movement_id and register.my_record = record.record_id; -- everything register

-- get name of people with bank accounts with debit cards associated to them
SELECT first_name, last_name from bank_account where account_number in (SELECT
my_account from credit_card where card_type = 'Debit');

-- bank_account index
CREATE index account_numbers on bank_account(account_number);
-- credit_card index
CREATE index card_numbers on credit_card(card_number);

-- View that shows what bank the people have
CREATE VIEW Bank AS select bank from bank_account;
```

6.) Indexes

In your system, indexes are used to accelerate searches and retrieval of specific bank account and credit card information based on the account_number and card_number columns, improving query performance. The queries are shown here:

CREATE index account_numbers on bank_account(account_number);

CREATE index card_numbers on credit_card(card_number);

7.) Views

In our system, a view is used to simplify data retrieval by abstracting complex table relationships into a reusable format. For example, the Bank view is created with the query:

CREATE VIEW Bank AS SELECT bank FROM bank_account;

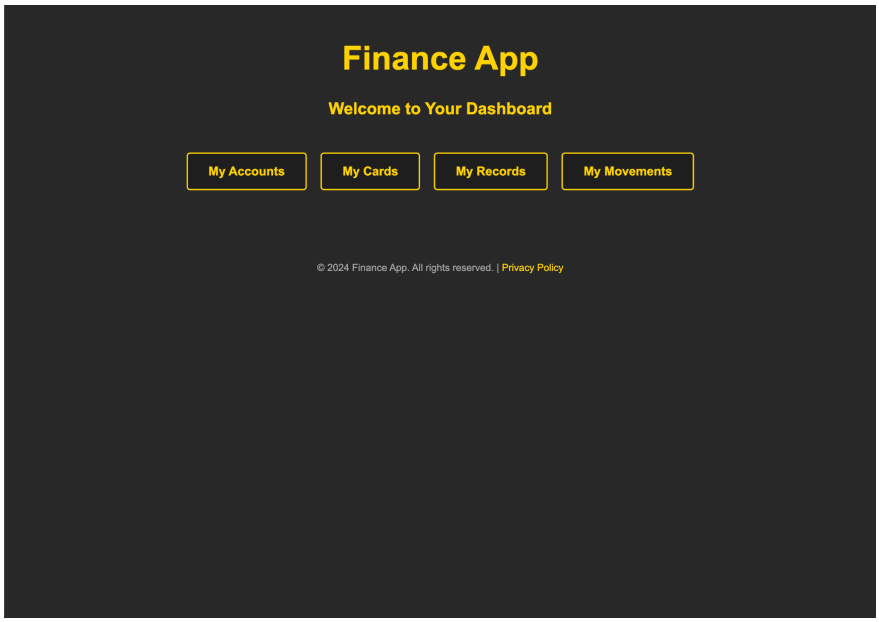
This view helps with abstraction and security, allowing users to access specific data (like the bank name) without needing to interact directly with the underlying bank_account table.

8.) Description of the Technology used for the interface

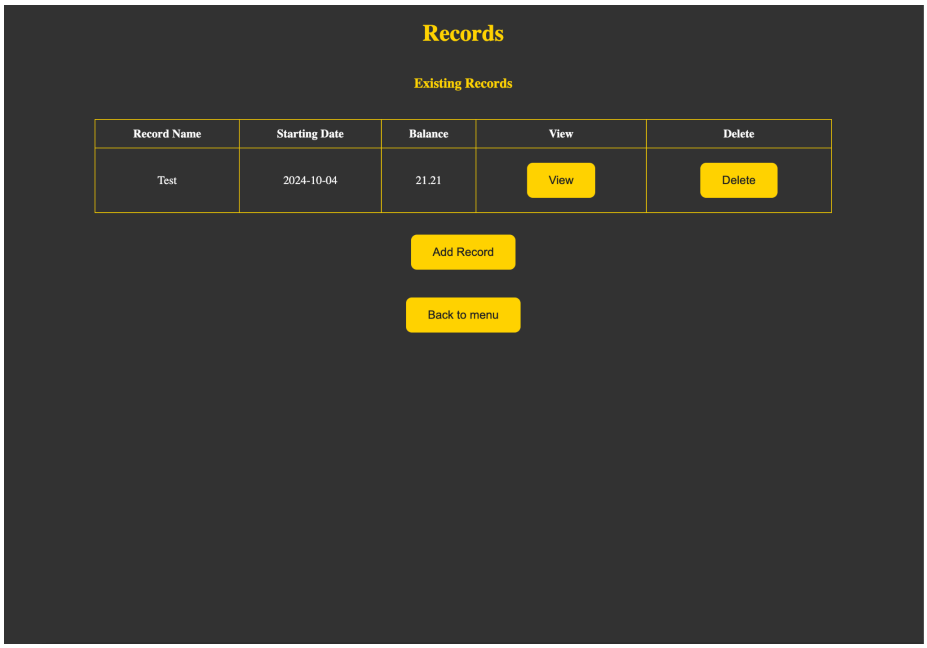
The interface for this project is implemented using Flask, a lightweight and versatile web framework for Python, enabling seamless integration of backend logic and frontend templates. HTML and CSS are used to create a responsive and visually appealing user interface, with features like styled forms and buttons for user interaction. The backend connects to a MySQL database for storing and retrieving financial data, leveraging structured queries to handle operations like inserts, updates, and deletions efficiently. The design prioritizes user accessibility and interactivity, ensuring a smooth experience for managing financial information.

9.) Screenshots of GUI

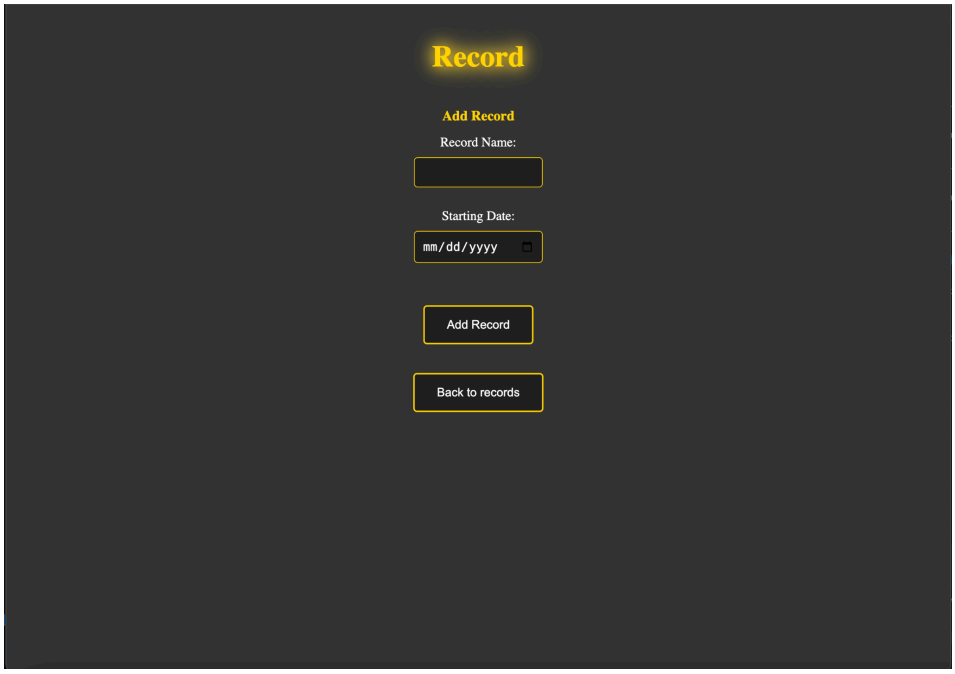
When the app is opened, the user it brought to the main menu:



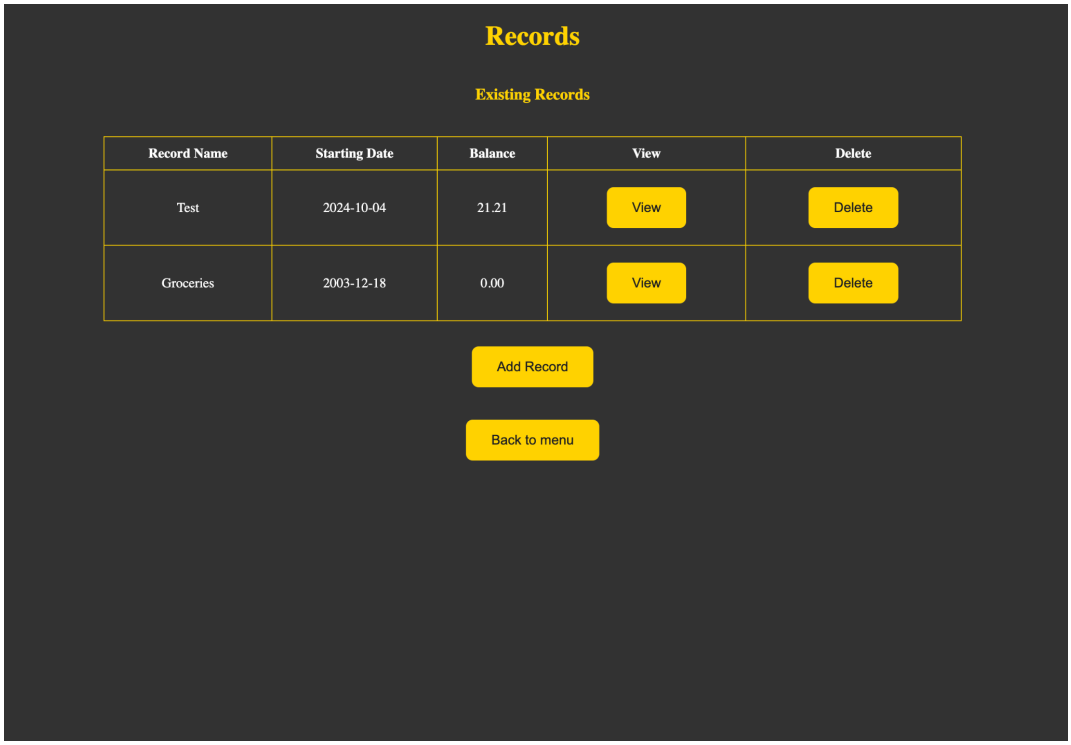
Accessing my records shows a record of the users balance changes where they can add or remove a record:



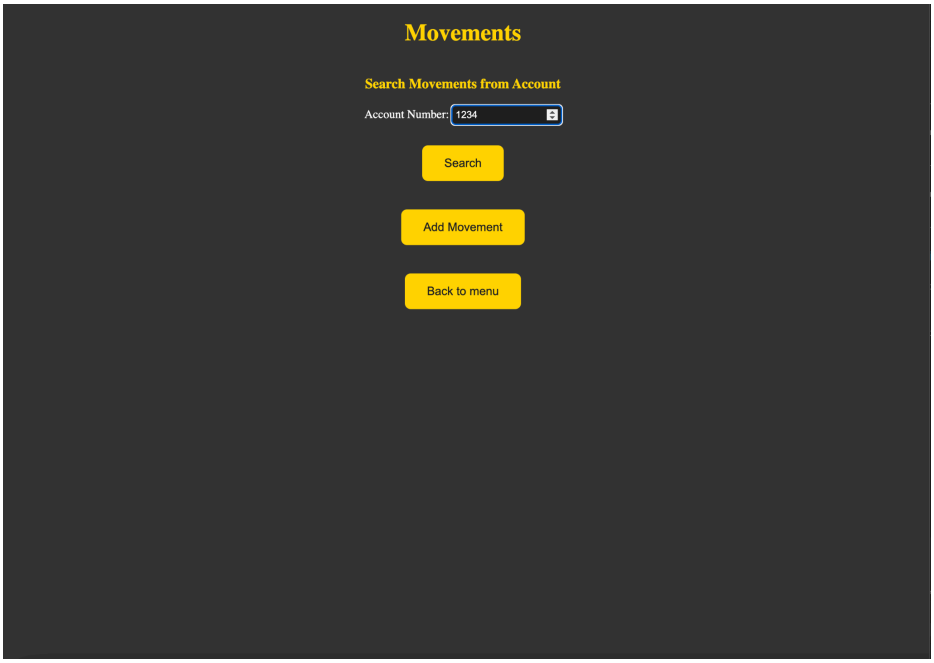
When a user wants to add a record, they can access this screen through the “Add Record” button:



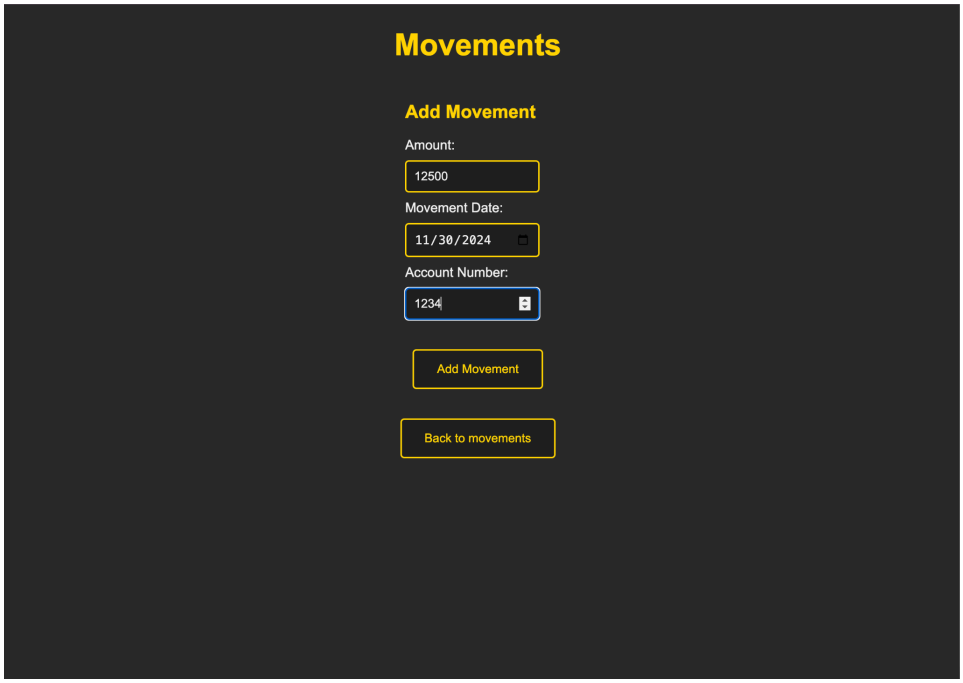
After a record is added, it is then displayed on the user’s list of records:



The movement menu features an option for searching an account and adding a movement:



The user may then enter the amount, date, and account number for the movement:



Then, the movement will be added to the database and will display on the user's movements: