

# **Banko**

Team 6

Ajit Jakasania, Gabriel Tenocelotl, Osama Hanhan

Professor : Dr. Mike Wu

CS157A - Fall 2020

# Table of Contents

Project Requirements Section	3
User Interaction	5
Project Design Section	5
Entity Relationship Diagram (ERD)	7
Relational Schema	7
Relationships	8
Entity and Relationship Explanation	8-10
MySQL Table Screenshots	11-21
Project Implementation	22-49
Procedure (Step by Step) of How to Set up and Run Website	49
Lesson Learned	49-50

# **Project Requirements Section:**

## **Functional Requirements**

- People need to create accounts - need to have sign up feature
  - Registration functionality for users to be able to create an account
  - Takes in a user's first name, last name, email, phone number, street name, zip code, city, country, username, password, and birthdate.
- Log in feature / log in page
  - Authenticates user with registration details; if correct details, redirects user to the product
  - Takes in a username and a password
- Generates visuals to show your spending patterns such as pie charts, trending graphs, etc.
  - Utilizes your transaction history to produce analysis of how you spend your money; allows users to learn from their spending
  - Utilizes your stored transaction history
- Different ways to split bills (flat amounts, percentages, different groups of roommates)
  - Allows flexibility in how you manage your transactions
  - Takes an input of how you'd like to split a transaction, whether it is per flat amount, or per percentage
- Database stores transactions, user data, banking data, permissions
  - This data is stored in MySQL to authenticate users, produce transaction data, check groups, etc.,
- Store receipts as proof of payment (png, jpeg, pdf)
  - Stores a link to a photo of a receipt, which allows users to store their receipts of the group transactions in a specific place, allowing them convenience and record keeping.
  - Takes in a URL for the photo
- Store bills for record keeping (png, jpeg, pdf)
  - Stores a link to a photo of the bill, allowing more intricate record keeping per transaction
  - Takes in a URL for the photo
- Tag bills with title, keywords, and date to be able to search them
  - Gives users flexibility and increases user experience
  - Search bar with options to choose from for certain keywords in the messages, or for a title that you can search, or by date
- Users can ask for payment from other users in their group
  - This allows users to tell groupmates that there is a new transaction that they must tend to
- Approval system for payments

- Allows the person who posted the transaction to confirm that people have made their payments
- Add a bill, remove a bill, update a bill
  - Flexibility and user experience when it comes to bill record-keeping
  - Takes a transaction and a URL
- Have a group-management page (shows all groups you're in, members of each group, shows how much each users owes you in each group and how much you owe them)
  - Gives the user quick data on what they may need to tend to
- Create new group
  - Allows users to make as many new groups as they would like
  - Takes in a group name and if it is unique, returns you a success message
- Users can request to join group by utilizing the unique group-id
  - A simple search-based group joining system to allow people to join groups quickly and easily
  - Sends the unique group name into the Join Group functionality
- Users in a group can accept another user into the group if that user requested to join
  - Users can approve a user who would like to join
  - Simple button click
- Users in a group with the right permissions can remove other users in that group
  - Users can remove other users in a group if they have the permissions to do so
  - Simple button click
- Have a home page (small summary of spending, a small feed of recent transactions, side bar with more options) information about all the groups you are in
  - A homepage that shows you any information you need to know; whether you owe money or your spending patterns; a summary page
- Have a Bills page (shows information about all the bills with comment section ordered by date)
  - Shows all the bills with the message appended to them
- Search feature / categorization feature in Bills page based on tag, date, title, etc
  - Gives users a more flexible interface to manage their records and expenses
  - Search bar with options to search the message, the title, or the date
- Have a Transactions page (Shows all the posts people made, whether each person paid their share, and the dates of when people paid)
  - Transactions page shows all the transactions you are a part of, and the information associated with that
- Page that shows user-specific data on their spending patterns in all groups
  - A summary page that tries to analyze your spending information
- Pay button for each bill will be both on homepage transaction feed and in transaction page (only if you have not paid already)
  - Allows a user to show that they have paid a bill
  - Simple button click
- Group page - Information about only a specific group, including transactions, bills and chat box.
  - Gives you information about a specific group, along with a group chatbox!

- Maybe auto-generate venmo requests when asking for payment? ~optional
  - An optional feature that allows users to generate a venmo request that they must confirm on their phones
- Email confirmation ~optional
  - Optional feature to send a confirmation through email to the users when their transaction is accepted or is paid
- Upload profile picture ~optional
  - Optional feature to allow profile pictures for the users

## User Interaction

Users would be able to interact with our application in multiple ways: they could make a personal group to manage their own expenses, and see trends in their transactions and use it for record keeping. Users can also utilize our application to upload shared bills with others and request and track payments. Users could also utilize our application to join a group that someone else had made, and access the bills and pay the transactions from there. All groups have a chat box, which allows easy communication between all the users of a group, and gives room for flexibility and clarity.

## Project Design Section:

### Graphical User Interface

The user interface will be minimalist, with rounded edges and easy to use features. This is an application that you should be able to open, track your finances, and leave. It should be as fast and easy to use as possible; while user retention is important, forcing users to spend extra time utilizing the application would only make it less likely for them to use it.

The home page, without being logged in, should have a graphic in the middle that states exactly what it is we are offering and examples of it. There should also be a ‘sign up now button’.

The sign up page and login page will look very similar, and should be extremely minimalist and plain.

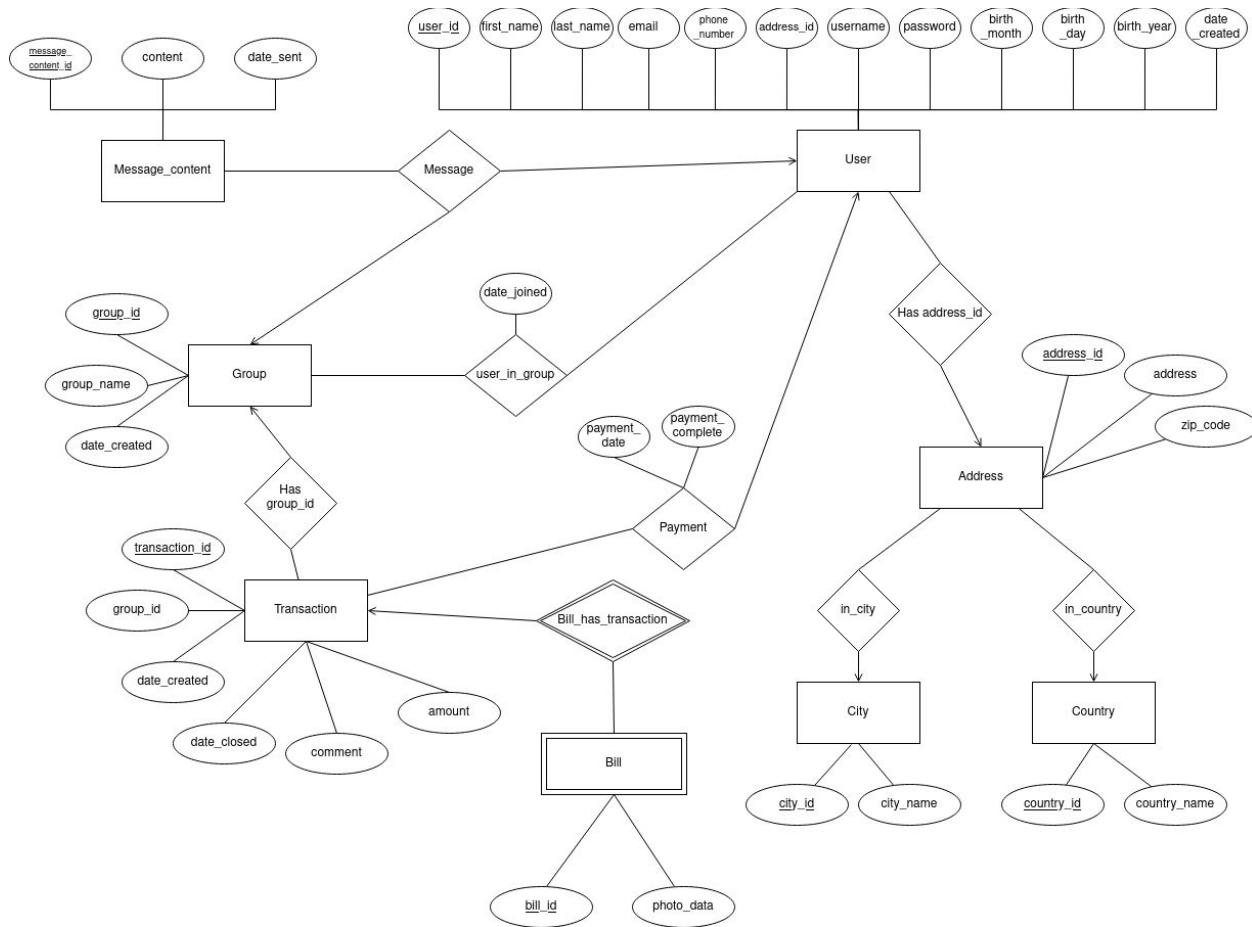
Once logged in, the home page will have a pi-chart showing your spending in the past month, with a list of pending-transactions, or if there are no pending-transactions, then there will be a ‘congratulations on paying everything!’ and then a list of your most recent paid off transactions with a green checkmark. There would be a sidebar with the options ‘bills’, ‘transactions’, ‘group mates’, and ‘switch groups’. This would have information about all the groups you are in

If you click on ‘bills’, then it would take you to a page that lists out each bill that was put into the system for that group, with an ability to expand the document (if there is a document that was submitted) alongside with the tag (bill, grocery, food,..) and the date uploaded and the date paid (if not paid yet, will say ‘unpaid’).

If you click on ‘transactions’, it would take you to a similar page as bills; same layout, same color scheme, but it is all the posts that people made about paying. Each of these posts will link to a bill, will have the tag, and the date made. It will also have who has paid and when. It will also link the actual post itself, which would have comments and whatnot.

If you click on ‘group-list’ it would show you each group you are in, with the name of the group and a tab that you can click to expand the information on that group. The information includes how much you owe/ are owed from each user in that group, and the members of that group.

## Entity Relationship Diagram (ERD)



## Relational Schema

user (user\_id, first\_name, last\_name, email, phone\_number, address\_id (f.k), username, hashed\_password, birth\_month, birth\_day, birth\_year, date\_created)  
address (address\_id, street\_name, zip\_code, city\_id (f.k), country\_id (f.k))  
country (country\_id, country\_name)  
city (city\_id, city\_name)  
group\_list (group\_id, group\_name, date\_created)  
user\_in\_group (user\_id (f.k), group\_id (f.k), date\_joined)  
transaction (transaction\_id, group\_id (f.k), date\_created, date\_closed, transaction\_content, amount)  
bill (bill\_id, transaction\_id, photo\_url)  
payment (user\_id (f.k), transaction\_id (f.k), payment\_date, payment\_complete)  
message\_content (message\_content\_id, content, date\_sent)  
message (message\_content\_id (f.k), group\_id (f.k), user\_id (f.k))

## Relationships

Relationship between entities	Left to Right	
user to address	Many	Exactly one
address to city	Many	Exactly one
address to country	Many	Exactly one
group to user	Many	Many
transaction to user	Many	Exactly one
transaction to group	Many	Exactly one
bill to transaction (weak relationship)	Many	Exactly one
message_content to group to user (3-way relationship)	Many : Exactly one : Exactly one	

## Entity and Relationship Explanation

### **User :**

The user entity holds information about each individual user. This information includes names, email addresses, phone numbers, and account information such as username and password. Address\_id is referenced in this table.

### **Address :**

The address entity contains the address information of each user. This data includes address\_id which is referenced to the user table, street\_name, zip\_code. The code\_id and country\_id is referenced in this table.

### **Country :**

The country entity contains the country\_id which is the primary key and auto increments. Each country\_id is mapped to a specific country so it can be used by the address entity for reference.

### **City :**

The city entity contains the city\_id which is the primary key and auto increments. Each city\_id is mapped to a specific city so it can be used by the address entity for reference.

### **Group\_list :**

The Group entity has the group\_id as the primary key and auto increments. It also contains the group\_name and date\_created (date and time).

### **User\_in\_group :**

The user\_in\_group relationship is a two way relationship between the user and group entities. The relationship is used to map each user to a group and the relationship is many to many meaning each user can be in multiple groups and two groups can contain the same user.

### **Transaction :**

The transaction entity holds information about each individual transaction. This information includes which group is the transaction from, the amount, what the transaction is about, and the date when the transaction was made and closed.

### **Bill :**

The bill is a weak entity where it relies on the entity transaction. The bill holds information about the photo of each transaction.

**Payment :**

The payment relationship is a two way relationship between user and transaction entities. The relationship also stores information such as when each payment towards the transaction is made. It also stores when the payment is complete.

**Message\_content :**

The message\_content entity has information about each message. This information is stored individually and has a date specified when sent. This entity has a relationship with the group and user entities.

**Message :**

The message relationship is a three way relationship between the message\_content, group and user entities. This is a many to exactly one to exactly one relationship because each message\_content is written exactly by one user to exactly one group. This means that one user can write many messages, but to only one group. A single group can contain many messages written by one user.

# MySQL Table Screenshots

## User:

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree, with 'omjmfm6vzmpqpc0p' selected. Under this schema, the 'Tables' node is expanded, showing 'address', 'bill', 'city', 'country', 'group\_list', 'message', 'message\_content', 'payment', 'transaction', 'user', and 'user\_in\_group'. The 'user' table is selected in the 'Tables' list. The main area shows the 'Result Grid' with 15 rows of data. The columns are: user\_id, first\_name, last\_name, email, phone\_number, address\_id, username, hashed\_password, birth\_month, birth\_day, birth\_year, and date\_created. The data includes various names like Osama, Gabe, Ajit, Test, etc., with corresponding email addresses and random hashed passwords.

user_id	first_name	last_name	email	phone_number	address_id	username	hashed_password	birth_month	birth_day	birth_year	date_created	
1	Osama	Hanhan	hanhan.osama@gmail.com	16505159552	1	user1	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:15:59
2	Gabe	XXXX	XX.GABE@gmail.com	14502153552	2	user2	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:19:15
3	Ajit	XXXX	XX.AjtE@gmail.com	14502153552	2	user3	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:19:15
4	Test	XXXX	XX.test1@gmail.com	14502153552	2	user4	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:19:15
5	Test346	XXXX	XX.test2@gmail.com	14502153552	2	user5	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:19:15
6	Test34226	XXXX	XX.test2@gmail.com	14502153552	2	user6	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:19:15
7	Guy1	XXXX	XX.5153@gmail.com	14502153552	3	user7	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:20:48
8	Guy12	XXXX	XX.23@gmail.com	14502153552	3	user8	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:20:48
9	Guy13	XXXX	XX.test11@gmail.com	14502153552	3	user9	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:20:48
10	Guy15	XXXX	XX.test25@gmail.com	14502153552	3	user10	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:20:48
11	Guy16	XXXX	XX.test42@gmail.com	14502153552	3	user11	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:20:48
12	Guy1124	XXXX	XX.5153@124mail.com	14502153552	8	user12	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:22:08
13	Guy12412	XXXX	XX.23@gmail.com	14502153552	8	user13	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:22:08
14	Guy1413	XXXX	XX.14test11@gmail.com	14502153552	8	user14	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:22:08
15	Guy1415	XXXX	XX.te1414st25@gmail.com	14502153552	8	user15	jhdsfgfdyhsjgjd..	128 chars LOL	2	14	1999	2020-10-31 00:22:08

user_id	first_name	last_name	email	phone_number	address_id	username	hashed_password	birth_month	birth_day	birth_year	date_created	
1	Osama	Hanhan	hanhan.osama@gmail.com	16505159552	1	user1	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:15:59
2	Gabe	XXXX	XX.GABE@gmail.com	14502153552	2	user2	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:19:15
3	Ajit	XXXX	XX.AjtE@gmail.com	14502153552	2	user3	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:19:15
4	Test	XXXX	XX.test1@gmail.com	14502153552	2	user4	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:19:15
5	Test346	XXXX	XX.test2@gmail.com	14502153552	2	user5	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:19:15
6	Test34226	XXXX	XX.test2@gmail.com	14502153552	2	user6	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:19:15
7	Guy1	XXXX	XX.5153@gmail.com	14502153552	3	user7	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:20:48
8	Guy12	XXXX	XX.23@gmail.com	14502153552	3	user8	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:20:48
9	Guy13	XXXX	XX.test11@gmail.com	14502153552	3	user9	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:20:48
10	Guy15	XXXX	XX.test25@gmail.com	14502153552	3	user10	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:20:48
11	Guy16	XXXX	XX.test42@gmail.com	14502153552	3	user11	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:20:48
12	Guy1124	XXXX	XX.5153@124mail.com	14502153552	8	user12	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:22:08
13	Guy12412	XXXX	XX.23@gmail.com	14502153552	8	user13	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:22:08
14	Guy1413	XXXX	XX.14test11@gmail.com	14502153552	8	user14	jhdsfgfdyhsjgjd..	128 chars LOL	5	14	1999	2020-10-31 00:22:08
15	Guy1415	XXXX	XX.te1414st25@gmail.com	14502153552	8	user15	jhdsfgfdyhsjgjd..	128 chars LOL	2	14	1999	2020-10-31 00:22:08

```
mysql> DESCRIBE user;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| user_id | int(10) unsigned | NO | PRI | NULL | auto_increment |
| first_name | varchar(32) | NO |   | NULL |           |
| last_name | varchar(32) | NO |   | NULL |           |
| email | varchar(64) | NO |   | NULL |           |
| phone_number | char(11) | YES |   | NULL |           |
| address_id | int(10) unsigned | NO | MUL | NULL |           |
| username | varchar(24) | NO |   | NULL |           |
| hashed_password | char(128) | NO |   | NULL |           |
| birth_month | tinyint(3) unsigned | NO |   | NULL |           |
| birth_day | tinyint(3) unsigned | NO |   | NULL |           |
| birth_year | smallint(5) unsigned | NO |   | NULL |           |
| date_created | datetime | NO |   | NULL |           |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
12 rows in set (0.09 sec)

mysql>
```

## Address:

The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane displays the 'SCHEMAS' section, with the 'omjmf6vzmpqpgc0p' schema selected. Under this schema, the 'Tables' section lists 'address', 'bill', 'city', 'country', 'group\_list', 'message', 'message\_content', 'payment', 'transaction', 'user', and 'user\_in\_group'. Below these are sections for 'Views', 'Stored Procedures', and 'Functions'. The 'Query 1' tab is active, showing the SQL query: 'SELECT \* FROM omjmf6vzmpqpgc0p.address;'. The 'Result Grid' shows the data for the 'address' table, which contains 15 rows of address information. The columns are: address\_id, street\_name, zip\_code, city\_id, and country\_id. The data includes various street names like Sesame Street, Buena Vista, and Hello Vista, with corresponding zip codes ranging from 75124 to 95126, and city and country IDs ranging from 1 to 8.

```
mysql> DESCRIBE address;
+-----+-----+-----+-----+-----+
| Field      | Type       | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+
| address_id | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| street_name | varchar(32)    | NO   |     | NULL    |                |
| zip_code   | varchar(16)    | YES  |     | NULL    |                |
| city_id    | int(10) unsigned | NO   | MUL | NULL    |                |
| country_id | int(10) unsigned | NO   | MUL | NULL    |                |
+-----+-----+-----+-----+-----+
5 rows in set (0.09 sec)

mysql>
```

## City :

The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane displays the schema 'omjmf6vzmpqpgc0p' with its tables: address, bill, city, country, group\_list, message, message\_content, payment, transaction, user, and user\_in\_group. The 'city' table is selected in the Query Editor tab.

**Query Editor:**

```
SELECT * FROM omjmf6vzmpqpgc0p.city;
```

**Result Grid:**

	city_id	city_name
▶	1	San Francisco
	2	New York
	3	Boston
	4	Washington DC
	5	Chicago
	6	London
	7	Budapest
	8	Paris
	9	Waterloo
	10	Mexico City
	11	Berlin
	12	Shanghai
	13	Austin
	14	Dallas
	15	Rio De Janeiro
*	NULL	NULL

**MySQL Command Line:**

```
mysql> DESCRIBE city;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| city_id | int(10) unsigned | NO | PRI | NULL | auto_increment |
| city_name | varchar(40) | NO | | NULL | |
+-----+-----+-----+-----+-----+
2 rows in set (0.10 sec)

mysql> 
```

## Country :

Navigator:.....

Query 1    country    X

SCHEMAS

Filter objects

omjmf6vzmpqpgc0p

Tables

- address
- bill
- city
- country
- group\_list
- message
- message\_content
- payment
- transaction
- user
- user\_in\_group

Views

Stored Procedures

Functions

Result Grid | Filter Rows: | Edit: |

	country_id	country_name
▶	1	United States
	2	Spain
	3	Canada
	4	Mexico
	5	Brazil
	6	Japan
	7	France
	8	Germany
	9	South Korea
	10	China
	11	Argentina
	12	Italy
	13	India
	14	Egypt
	15	Philippines
*	NULL	NULL

Administration   Schemas   Information

```
mysql> DESCRIBE country;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| country_id | int(10) unsigned | NO | PRI | NULL | auto_increment |
| country_name | varchar(24) | NO | | NULL | |
+-----+-----+-----+-----+-----+
2 rows in set (0.09 sec)
```

```
mysql> |
```

## Group\_list

Navigator : Query 1 group\_list ×

SCHEMAS Filter objects

omjmf6vzmpqpgc0p

Tables address bill city country group\_list message message\_content payment transaction user user\_in\_group

Views

Stored Procedures

Functions

Administration Schemas

Information

No object selected

Query 1 group\_list ×

1 • SELECT \* FROM omjmf6vzmpqpgc0p.group\_list;

Result Grid | Filter Rows: Edit:

	group_id	group_name	date_created
▶	1	Roommates 328	2020-10-31 00:14:39
	2	The Meme Team	2020-10-31 00:15:47
	3	All the Single Ladies	2020-10-31 00:16:14
	4	The Three Musketeers	2020-10-31 00:16:41
	5	Fantastic 4	2020-10-31 00:17:51
	6	Dream Team	2020-10-31 00:18:13
	7	Full House	2020-10-31 00:19:28
	8	Pearfect Roommates	2020-10-31 00:20:02
	9	The Fab Five	2020-10-31 00:20:32
	10	Game of Phones	2020-10-31 00:20:52
	11	The Trouble Makers	2020-10-31 00:22:52
	12	Best Buddies	2020-10-31 00:23:48
	13	The Dreamers	2020-10-31 00:24:16
	14	Happy Home	2020-10-31 00:26:33
	15	Family	2020-10-31 00:27:59
*	NULL	NULL	NULL

mysql> DESCRIBE group\_list;

Field	Type	Null	Key	Default	Extra
group_id	int(10) unsigned	NO	PRI	NULL	auto_increment
group_name	varchar(32)	NO		NULL	
date_created	datetime	NO		NULL	

3 rows in set (0.09 sec)

mysql>

### User\_in\_group :

The screenshot shows the MySQL Workbench interface. The left pane displays the schema structure for the database 'omjmf6vzmpqpgc0p'. The 'Tables' section lists various tables including 'address', 'bill', 'city', 'country', 'group\_list', 'message', 'message\_content', 'payment', 'transaction', 'user', and 'user\_in\_group'. The 'user\_in\_group' table is selected. The right pane shows the results of the query 'SELECT \* FROM omjmf6vzmpqpgc0p.user\_in\_group;'. The results are presented in a grid format with columns: user\_id, group\_id, and date\_joined. The data shows 15 rows of user-group associations, all joined on 2020-10-31 00:27:08.

	user_id	group_id	date_joined
▶	1	1	2020-10-31 00:27:07
	2	1	2020-10-31 00:27:07
	3	1	2020-10-31 00:27:07
	4	1	2020-10-31 00:27:07
	5	1	2020-10-31 00:27:08
	6	2	2020-10-31 00:27:08
	7	2	2020-10-31 00:27:08
	8	2	2020-10-31 00:27:08
	9	2	2020-10-31 00:27:08
	10	2	2020-10-31 00:27:08
	11	6	2020-10-31 00:27:08
	12	8	2020-10-31 00:27:08
	13	10	2020-10-31 00:27:08
	14	14	2020-10-31 00:27:08
	15	14	2020-10-31 00:28:00

```
mysql> DESCRIBE user_in_group
      -> ;
+-----+-----+-----+-----+-----+
| Field      | Type           | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| user_id    | int(10) unsigned | NO   | MUL | NULL    |       |
| group_id   | int(10) unsigned | NO   | MUL | NULL    |       |
| date_joined | datetime        | NO   |       | NULL    |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.08 sec)
```

```
mysql>
```

## Transaction :

Navigator transaction

**SCHEMAS**

Filter objects

**omjmf6vzmpqpgc0p**

- Tables
  - address
  - bill
  - city
  - country
  - group\_list
  - message
  - message\_content
  - payment
  - transaction
  - user
  - user\_in\_group
- Views
- Stored Procedures
- Functions

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap:

	transaction_id	group_id	date_created	date_closed	transaction_content	amount
▶	1	1	2020-10-31 00:30:17	0000-00-00 00:00:00	PAY ME RENT	0.00
	2	1	2020-10-31 00:30:17	0000-00-00 00:00:00	PAY ME RENT	0.00
	3	1	2020-10-31 00:30:17	0000-00-00 00:00:00	PAY ME RENT	0.00
	4	1	2020-10-31 00:30:17	0000-00-00 00:00:00	PAY ME RENT	0.00
	5	1	2020-10-31 00:30:18	0000-00-00 00:00:00	PAY ME RENT	0.00
	6	1	2020-10-31 00:30:18	0000-00-00 00:00:00	PAY ME RENT	0.00
	7	4	2020-10-31 00:30:18	0000-00-00 00:00:00	PAY ME FOOD	0.00
	8	4	2020-10-31 00:30:18	0000-00-00 00:00:00	PAY ME RENT	0.00
	9	4	2020-10-31 00:30:18	0000-00-00 00:00:00	PAY ME RENT	0.00
	10	4	2020-10-31 00:30:18	0000-00-00 00:00:00	PAY ME RENT	0.00
	11	8	2020-10-31 00:30:19	0000-00-00 00:00:00	PAY ME PGE	0.00
	12	4	2020-10-31 00:30:19	0000-00-00 00:00:00	PAY ME RENT	0.00
	13	2	2020-10-31 00:30:19	0000-00-00 00:00:00	PAY ME DOG	0.00
	14	3	2020-10-31 00:30:19	0000-00-00 00:00:00	PAY ME CAT	0.00
	15	6	2020-10-31 00:37:29	2020-10-31 00:37:39	i love u	100.69
*	HULL	HULL	HULL	HULL	HULL	HULL

No object selected

```
mysql> DESCRIBE transaction;
```

Field	Type	Null	Key	Default	Extra
transaction_id	int(10) unsigned	NO	PRI	NULL	auto_increment
group_id	int(10) unsigned	NO	MUL	NULL	
date_created	datetime	NO		NULL	
date_closed	datetime	NO		NULL	
transaction_content	varchar(128)	YES		NULL	
amount	double(8,2)	NO		NULL	

6 rows in set (0.09 sec)

## Bill :

The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane displays the 'SCHEMAS' section, with 'omjmf6vzmpqpgc0p' selected. Under this schema, the 'Tables' section lists 'address', 'bill', 'city', 'country', 'group\_list', 'message', 'message\_content', 'payment', 'transaction', 'user', and 'user\_in\_group'. Below these are sections for 'Views', 'Stored Procedures', and 'Functions'. At the bottom of the Navigator are tabs for 'Administration' and 'Schemas', with 'Information' also visible.

The main area contains a 'Query 1' tab with the query `SELECT * FROM omjmf6vzmpqpgc0p.bill;`. To the right of the query is a toolbar with various icons. Below the toolbar is a result grid titled 'Result Grid' with columns: bill\_id, transaction\_id, and photo\_url. The data in the grid is as follows:

	bill_id	transaction_id	photo_url
1	15		<a href="https://www.google.com/search?q=bill&amp;source...">https://www.google.com/search?q=bill&amp;source...</a>
2	1		<a href="https://www.google.com/search?q=bill&amp;source...">https://www.google.com/search?q=bill&amp;source...</a>
3	14		<a href="https://www.google.com/search?q=bill&amp;source...">https://www.google.com/search?q=bill&amp;source...</a>
4	13		<a href="https://www.google.com/search?q=bill&amp;source...">https://www.google.com/search?q=bill&amp;source...</a>
5	12		<a href="https://www.google.com/search?q=bill&amp;source...">https://www.google.com/search?q=bill&amp;source...</a>
6	4		<a href="https://www.google.com/search?q=bill&amp;source...">https://www.google.com/search?q=bill&amp;source...</a>
7	2		<a href="https://www.google.com/search?q=bill&amp;source...">https://www.google.com/search?q=bill&amp;source...</a>
8	3		<a href="https://www.google.com/search?q=bill&amp;source...">https://www.google.com/search?q=bill&amp;source...</a>
9	9		<a href="https://www.google.com/search?q=bill&amp;source...">https://www.google.com/search?q=bill&amp;source...</a>
10	5		<a href="https://www.google.com/search?q=bill&amp;source...">https://www.google.com/search?q=bill&amp;source...</a>
11	8		<a href="https://www.google.com/search?q=bill&amp;source...">https://www.google.com/search?q=bill&amp;source...</a>
12	6		<a href="https://www.google.com/search?q=bill&amp;source...">https://www.google.com/search?q=bill&amp;source...</a>
13	7		<a href="https://www.google.com/search?q=bill&amp;source...">https://www.google.com/search?q=bill&amp;source...</a>
14	10		<a href="https://www.google.com/search?q=bill&amp;source...">https://www.google.com/search?q=bill&amp;source...</a>
15	11		<a href="https://www.google.com/search?q=bill&amp;source...">https://www.google.com/search?q=bill&amp;source...</a>

```
mysql> DESCRIBE bill;
+-----+-----+-----+-----+-----+
| Field      | Type       | Null | Key | Default | Extra       |
+-----+-----+-----+-----+-----+
| bill_id    | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| transaction_id | int(10) unsigned | NO   | MUL | NULL    |               |
| photo_url  | varchar(512)    | NO   |     | NULL    |               |
+-----+-----+-----+-----+-----+
3 rows in set (0.09 sec)

mysql>
```

## Payment :

Navigator.....

Query 1 payment x

SCHEMAS  
Filter objects

omjmf6vzmpqpgc0p  
Tables  
address  
bill  
city  
country  
group\_list  
message  
message\_content  
payment  
transaction  
user  
user\_in\_group  
Views  
Stored Procedures  
Functions

Result Grid | Filter Rows: Export: Wrap

	user_id	transaction_id	payment_date	payment_complete
▶	1	1	2020-10-31 00:41:27	0
	2	2	2020-10-31 00:41:27	1
	3	3	2020-10-31 00:41:27	0
	4	4	2020-10-31 00:41:27	0
	5	5	2020-10-31 00:41:27	1
	6	6	2020-10-31 00:41:27	0
	7	7	2020-10-31 00:41:27	0
	8	8	2020-10-31 00:41:27	1
	9	9	2020-10-31 00:41:27	0
	10	10	2020-10-31 00:41:27	0
	11	11	2020-10-31 00:41:27	0
	12	12	2020-10-31 00:41:28	1
	13	13	2020-10-31 00:41:28	0
	14	14	2020-10-31 00:41:28	1
	15	15	2020-10-31 00:41:28	0

Information.....

No object selected

payment 1 x

```
mysql> DESCRIBE payment;
+-----+-----+-----+-----+-----+
| Field          | Type           | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| user_id        | int(10) unsigned | NO   | MUL | NULL    |
| transaction_id | int(10) unsigned | NO   | MUL | NULL    |
| payment_date   | datetime        | YES  |      | NULL    |
| payment_complete| tinyint(1)     | YES  |      | NULL    |
+-----+-----+-----+-----+-----+
4 rows in set (0.08 sec)

mysql>
```

## Message\_content :

Schemas

Filter objects

**omjmf6vzmpqpgc0p**

- Tables
  - address
  - bill
  - city
  - country
  - group\_list
  - message
  - message\_content
  - payment
  - transaction
  - user
  - user\_in\_group
- Views
- Stored Procedures
- Functions

Administration Schemas

No object selected

Query 1 message\_content

SELECT \* FROM omjmf6vzmpqpgc0p.message\_content;

message_content_id	content	date_sent
1	Osama is you forgot to pay October rent	2020-10-31 00:20:03
2	It happens	2020-10-31 00:20:31
3	No it doesn't pay your bills on time	2020-10-31 00:20:45
4	Even Ajit agrees that you should pay ...	2020-10-31 00:21:34
5	What do mean Ajit said that?!	2020-10-31 00:22:23
6	You can ask him yourself	2020-10-31 00:22:48
7	Hey Osama do you remember what as...	2020-10-31 00:24:14
8	I don't	2020-10-31 00:24:24
9	I remember the assigment its on thre...	2020-10-31 00:26:05
10	Thank you Ajit!	2020-10-31 00:26:28
11	Hey guys I think we messed up how t...	2020-10-31 00:27:33
12	What do you mean we messed up?	2020-10-31 00:27:45
13	Yea we messed up the total amount b...	2020-10-31 00:28:54
14	Ok we can cancel last transaction and ...	2020-10-31 00:29:43
15	Sounds perfect thanks Osama!	2020-10-31 00:30:00
*	HULL	HULL

```
mysql> DESCRIBE message_content;
+-----+-----+-----+-----+-----+
| Field          | Type           | Null | Key | Default | Extra       |
+-----+-----+-----+-----+-----+
| message_content_id | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| content         | varchar(512)    | NO   |     | NULL    |             |
| date_sent       | datetime        | NO   |     | NULL    |             |
+-----+-----+-----+-----+-----+
3 rows in set (0.09 sec)

mysql> 
```

## Message :

Navigator message

Query 1 message

1 • SELECT \* FROM omjmf6vzmpqpgc0p.message;

Result Grid | Filter Rows: Export:

	message_content_id	group_id	user_id
▶	1	1	2
	2	1	1
	3	1	2
	4	1	3
	5	1	1
	6	1	2
	7	3	2
	8	3	1
	9	3	3
	10	3	2
	11	8	6
	12	8	5
	13	8	7
	14	8	6
	15	8	5

Administration Schemas

Information

The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane displays the 'SCHEMAS' section, with 'omjmf6vzmpqpgc0p' selected. Under this schema, the 'Tables' section lists 'address', 'bill', 'city', 'country', 'group\_list', 'message', 'message\_content', 'payment', 'transaction', 'user', and 'user\_in\_group'. Below these are 'Views', 'Stored Procedures', and 'Functions'. The 'message' table is selected in the 'Tables' list. The main area is titled 'Query 1 message' and contains the SQL query 'SELECT \* FROM omjmf6vzmpqpgc0p.message;'. To the right of the query is a 'Result Grid' showing the data from the 'message' table. The grid has columns for 'message\_content\_id', 'group\_id', and 'user\_id'. The data consists of 15 rows, each with values for these three columns. At the bottom of the interface, there are tabs for 'Administration' and 'Schemas', and a 'Information' section.

```
mysql> DESCRIBE message;
+-----+-----+-----+-----+-----+
| Field          | Type           | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| message_content_id | int(10) unsigned | NO   | MUL | NULL    |       |
| group_id        | int(10) unsigned | NO   | MUL | NULL    |       |
| user_id         | int(10) unsigned | NO   | MUL | NULL    |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.09 sec)

mysql> 
```

## Project Implementation Section:

### **Register Functionality:**

Here is our frontend for our register page.

The screenshot shows a web browser window titled "React App" with the URL "localhost:3000/Register". The page has a header with "Home", "Log In", and "Register" buttons. The main content area is titled "Register" and contains a form with the following fields:

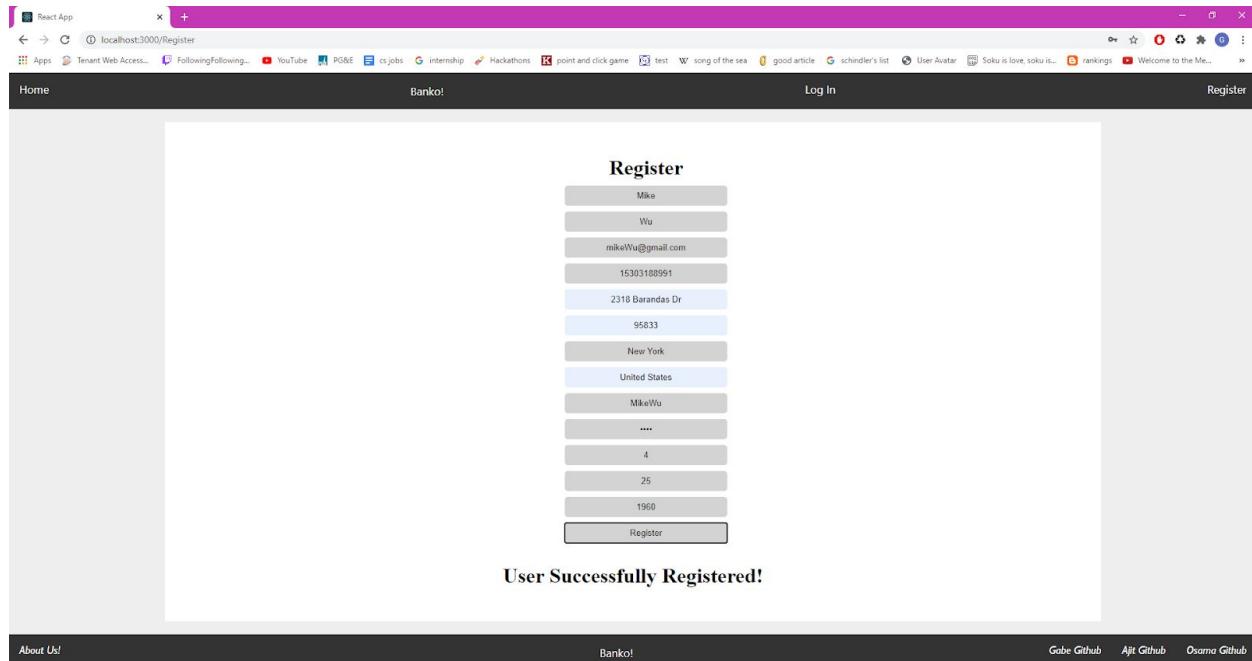
- First Name
- Last name
- email
- Phone Number
- Street Name
- Zipcode
- City
- Country
- Username
- Password
- Birth Month
- Birth Day
- Birth Year
- Register

Here is the Users Table before we register a new user.

The screenshot shows a database table named "Result Grid" with the following columns: user\_id, first\_name, last\_name, email, phone\_number, address\_id, username, hashed\_password, birth\_month, birth\_day, birth\_year, and date\_created. The data in the table is as follows:

user_id	first_name	last_name	email	phone_number	address_id	username	hashed_password	birth_month	birth_day	birth_year	date_created
22	Gabriel	Tenocelot	gabrieltencelot@gmail.com	15303078814	24	gabe	\$25	1	30	1997	2020-11-30 18:22:00
23	Gabriel	Tenocelot	gabrieltencelot@gmail.com	15303078814	25	hihi	\$25	1	1	1	2020-11-30 20:18:00
24	Gabriel	Tenocelot	gabrieltencelot@gmail.com	53030788141	26	something	\$25	1	1	1	2020-11-30 20:19:36
25	Gabriel	Tenocelot	gabrieltencelot@gmail.com	15303078814	27	asdasd	\$25	1	1	1	2020-11-30 20:22:45
26	Gabriel	Tenocelot	gabrieltencelot@gmail.com	53030788141	28	sfdsdfds	\$25	1	1	1	2020-11-30 20:24:38
27	Gabriel	Tenocelot	gabrieltencelot@gmail.com	53030788141	29	asdasdew...	\$25	1	1	1	2020-11-30 20:27:22
28	Gabriel	Tenocelot	gabrieltencelot@gmail.com	53030788141	30	123123124	\$25	1	1	1	2020-11-30 20:32:48
29	Gabriel	Tenocelot	gabrieltencelot@gmail.com	53030788141	31	irock194	\$25	1	1	1	2020-11-30 20:50:00
30	John	Jones	john.jones@gmail.com	16692965372	32	johnj	\$25	7	26	1999	2020-12-01 19:32:02
31	Osama	Hanhan	osama.hanhan@sjtu.edu	NULL	33	osama12	\$25	5	14	1999	2020-12-02 00:43:45
32	Gabe	Osama	email	NULL	34	gabeOsa...	\$25	1	1	1	2020-12-02 01:37:37
33	Jo	Jo	jo	NULL	35	sssss	\$25	1	1	1	2020-12-03 08:58:31
34	Jo	Jo	jo	NULL	36	ssssss	\$25	1	1	1	2020-12-03 08:58:52
35	Jo	Jo	jo	NULL	37	ssssss	\$25	1	1	1	2020-12-03 08:59:50
36	Roshdy	Hanhan	hsddb	NULL	38	roshdy	\$25	1	1	1	2020-12-03 12:36:38
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Here is the information we are inserting from the frontend



Here is the database after we registered the new user.

user_id	first_name	last_name	email	phone_number	address_id	username	hashed_password	birth_month	birth_day	birth_year	date_created
23	Gabriel	Tenocelotl	gabrieltenocelotl@gmail.com	15303078814	25	hihi	KL	1	1	1	2020-11-30 20:18:00
24	Gabriel	Tenocelotl	gabrieltenocelotl@gmail.com	53030788141	26	something	KL	1	1	1	2020-11-30 20:19:36
25	Gabriel	Tenocelotl	Tenocelotl	15303078814	27	asdasd	KL	1	1	1	2020-11-30 20:22:45
26	Gabriel	Tenocelotl	gabrieltenocelotl@gmail.com	53030788141	28	sfdssdfs	KL	1	1	1	2020-11-30 20:24:38
27	Gabriel	Tenocelotl	gabrieltenocelotl@gmail.com	53030788141	29	asdasdw...	KL	1	1	1	2020-11-30 20:27:22
28	Gabriel	Tenocelotl	gabrieltenocelotl@gmail.com	53030788141	30	123123124	KL	1	1	1	2020-11-30 20:32:48
29	Gabriel	Tenocelotl	gabrieltenocelotl@gmail.com	53030788141	31	irock194	SL	1	1	1	2020-11-30 20:50:00
30	John	Jones	john.jones@gmail.com	16692965372	32	johnj	SL	7	26	1999	2020-12-01 19:32:02
31	Osama	Hanhan	osama.hanhan@sjsu.edu	NULL	33	osama12	SL	5	14	1999	2020-12-02 00:43:45
32	Gabe	Osama	email	NULL	34	gabeOsa...	SL	1	1	1	2020-12-02 01:37:37
33	Jo	Jo	jo	NULL	35	ssss	SL	1	1	1	2020-12-03 08:58:31
34	Jo	Jo	jo	NULL	36	sssss	SL	1	1	1	2020-12-03 08:58:52
35	Jo	Jo	jo	NULL	37	ssssss	SL	1	1	1	2020-12-03 08:59:50
36	Roshdy	Hanhan	hsddb	NULL	38	roshdy	SL	1	1	1	2020-12-03 12:36:38
37	Mike	Wu	mikeWu@gmail.com	15303188991	39	MikeWu	SL	4	25	1960	2020-12-11 02:40:42
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

As you can see the new user with the correct information has been updated to the database.

Here is the code to register the User along with the query. This is the User.java file.

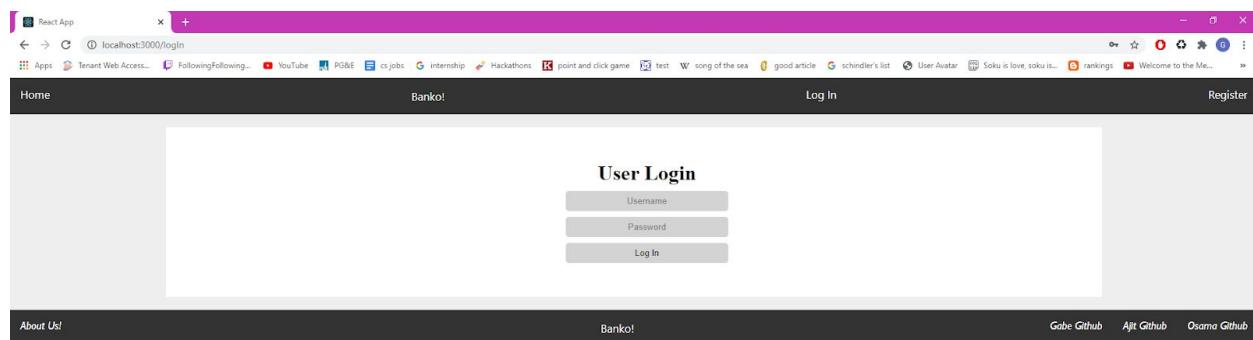
```
163 public int registerUser() throws SQLException {
164
165     int status = 0;
166     Connection connection = BancoBackendServer.connection;
167
168     if (phone_number == null || phoneNumberDigits(this.phone_number)) { // phonenumer is optional
169         if (uniqueUsername(this.username)) {
170
171             // gets our country and city ids, and then generates our address id.
172             int country_id = getCountryId(country_name, connection);
173             int city_id = getCityId(city_name, connection);
174
175             // if zip empty, -1 sent to createAddressID
176             int address_id = createAddressID(country_id, city_id, street_name, zip_code, connection);
177
178             String hashed_password = hashString(this.hashed_password);
179
180             // if phone number given, cool, if not, cool
181             String query;
182             if (phone_number != null) {
183                 query = "INSERT INTO omjmff6vzmpqpgc0p.user (first_name, last_name, "
184                     + "email, phone_number, address_id, username, hashed_password, birth_month, birth_day, birth_year, date_created) VALUES "
185                     + "(" + first_name + ", " + last_name + ", " + email + ", " + phone_number + ", "
186                     + address_id + ", " + username + ", " + hashed_password + ", " + birth_month + ", "
187                     + birth_day + ", " + birth_year + ", NOW())";
188             } else {
189                 query = "INSERT INTO omjmff6vzmpqpgc0p.user (first_name, last_name, "
190                     + "email, address_id, username, hashed_password, birth_month, birth_day, birth_year, date_created) VALUES "
191                     + "(" + first_name + ", " + last_name + ", " + email + ", " + address_id + ", "
192                     + username + ", " + hashed_password + ", " + birth_month + ", " + birth_day + ", "
193                     + birth_year + ", NOW())";
194             }
195
196             try {
197                 Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
198                     ResultSet.CONCUR_UPDATABLE);
199
200                 statement.executeUpdate(query);
201
202                 status = 3;
203             } catch (SQLException e) {
204                 e.printStackTrace();
205             }
206         }
207     }
208 }
```

The information is being gathered from the UserRegistrationController.java where we are creating a new User object.

```
66     @PostMapping(value = "/userRegister")
67     public @ResponseBody String addNewUser(@RequestBody User user) {
68         int status = 0;
69         try {
70             status = user.registerUser();
71         } catch (SQLException throwables) {
72             throwables.printStackTrace();
73         }
74         if (status == 1)
75             return "Failed as phoneNumber is not 11 digits!";
76         else if (status == 2)
77             return "Failed as username is not unique!";
78         else if (status == 3)
79             return "User Successfully Registered!";
80         return "Check the userRegister post method (backend)";
81     }
82
83 }
84
```

## Login Functionality:

Here is the frontend design of our Login page.



Here is the database with the previously stored User information

user_id	first_name	last_name	email	phone_number	address_id	username	hashed_password	birth_month	birth_day	birth_year	date_created
23	Gabriel	Tenocelot	gabrieltencelot@gmail.com	15303078814	25	hihi	KL	1	1	1	2020-11-30 20:18:00
24	Gabriel	Tenocelot	gabrieltencelot@gmail.com	53030788141	26	something	1	1	1	1	2020-11-30 20:19:36
25	Gabriel	Tenocelot	gabrieltencelot@gmail.com	15303078814	27	asdasd	1	1	1	1	2020-11-30 20:22:45
26	Gabriel	Tenocelot	gabrieltencelot@gmail.com	53030788141	28	sfdsdfds	1	1	1	1	2020-11-30 20:24:38
27	Gabriel	Tenocelot	gabrieltencelot@gmail.com	53030788141	29	asdasdw...	1	1	1	1	2020-11-30 20:27:22
28	Gabriel	Tenocelot	gabrieltencelot@gmail.com	53030788141	30	123123124	1	1	1	1	2020-11-30 20:32:48
29	Gabriel	Tenocelot	gabrieltencelot@gmail.com	53030788141	31	irock194	Šřš	1	1	1	2020-11-30 20:50:00
30	John	Jones	john.jones@mail.com	16692965372	32	johnj	Šřš	7	26	1999	2020-12-01 19:32:02
31	Osama	Hanhan	osama.hanhan@sjsu.edu	NULL	33	osama12	Šřš	5	14	1999	2020-12-02 00:43:45
32	Gabe	Osama	email	NULL	34	gabeOsa...	Šřš	1	1	1	2020-12-02 01:37:37
33	Jo	Jo	jo	NULL	35	ssss	Šřš	1	1	1	2020-12-03 08:58:31
34	Jo	Jo	jo	NULL	36	sssss	Šřš	1	1	1	2020-12-03 08:58:52
35	Jo	Jo	jo	NULL	37	ssssss	Šřš	1	1	1	2020-12-03 08:59:50
36	Roshdy	Hanhan	hsddb	NULL	38	roshdy	Šřš	1	1	1	2020-12-03 12:36:38
37	Mike	Wu	mikeWu@gmail.com	15303188991	39	MikeWu	Šřš	4	25	1960	2020-12-11 02:40:42
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

We enter the username and password we registered with.

The screenshot shows a browser window with the title 'React App'. The address bar shows 'localhost:3000/login'. The main content area displays a 'User Login' form. It has two input fields: one for 'username' containing 'MikeWu' and another for 'password' which is obscured by dots. Below the password field is a 'Log In' button. At the bottom of the page, there are links for 'About Us!', 'Banko!', 'Gabe Github', 'Ajit Github', and 'Osama Github'.

When we enter the correct username and password we are redirected to the homepage for logged in users.

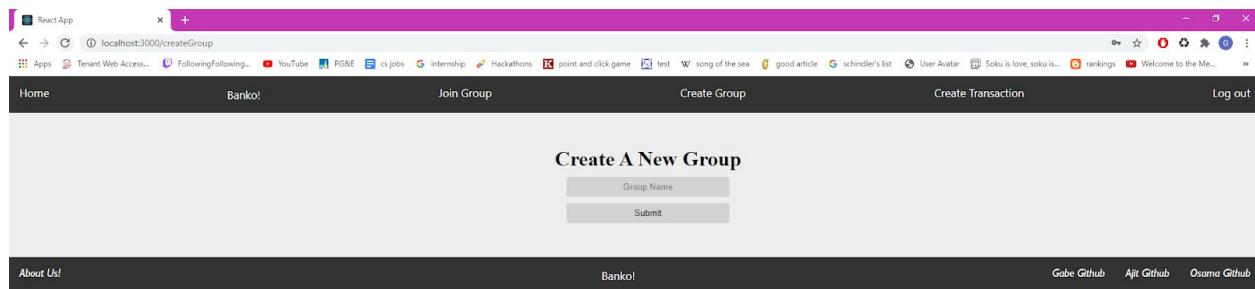
The screenshot shows a browser window with the title 'React App'. The address bar shows 'localhost:3000'. The main content area displays a homepage for logged-in users. The top navigation bar includes 'Home', 'Banko!', 'Join Group', 'Create Group', 'Create Transaction', and 'Log out'. On the left, there is a sidebar with links for 'Groups', 'Messages', and 'Transactions'. The main content area contains several promotional messages: 'Manage Your Expenses Interactively!', 'Banking With Friends!', 'See Your Transaction History!', 'Utilize a Message Board for Your Financial Needs!', and multiple instances of 'View Your Spending Patterns!'. At the bottom, there are links for 'About Us!', 'Banko!', 'Gabe Github', 'Ajit Github', and 'Osama Github'.

This is called when a user tries to log in; it gets the user's hashed password, and compares it to the hashed inserted password, then returns a boolean value.

```
public static boolean authenticatePassword(String username, String password) {  
    String tempPw = hashString(password);  
    Connection connection = BankoBackendServer.connection;  
  
    String query = "SELECT hashed_password FROM omjmf6vzmpqgc0p.user WHERE username = '" + username + "'";  
  
    try {  
        Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,  
            ResultSet.CONCUR_UPDATABLE);  
  
        ResultSet rs = statement.executeQuery(query);  
  
        // if they are equal, return true  
        if (rs.next())  
            if (rs.getString("hashed_password").equals(tempPw))  
                return true;  
  
        return false;  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
    return false;  
}
```

## Creating Group Functionality:

Here is the frontend for creating a new group page. You can click on the navbar to navigate to the create group page.



Here is the database before we create a new group.

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' tree view is expanded to show the 'omjmf6vzmpqpgc0p' schema, which contains tables like address, bill, city, country, group\_list, message, message\_content, payment, transaction, user, and user\_in\_group. Below the schema tree is a 'Views', 'Stored Procedures', and 'Functions' section. The main area displays the 'Result Grid' for the 'group\_list' table. The table has columns: group\_id, group\_name, and date\_created. The data grid shows 18 rows of group names and their creation dates, starting from group\_id 4 ('The Three Musketeers') and ending with group\_id 18 ('CS157A').

group_id	group_name	date_created
4	The Three Musketeers	2020-10-31 00:16:41
5	Fantastic 4	2020-10-31 00:17:51
6	Dream Team	2020-10-31 00:18:13
7	Full House	2020-10-31 00:19:28
8	Pearfect Roommates	2020-10-31 00:20:02
9	The Fab Five	2020-10-31 00:20:32
10	Game of Phones	2020-10-31 00:20:52
11	The Trouble Makers	2020-10-31 00:22:52
12	Best Buddies	2020-10-31 00:23:48
13	The Dreamers	2020-10-31 00:24:16
14	Happy Home	2020-10-31 00:26:33
15	Family	2020-10-31 00:27:59
16	Team 7	2020-12-01 00:40:42
17	The best coders	2020-12-01 05:33:17
18	CS157A	2020-12-02 01:38:28

Here is the frontend when we create a new group.

The screenshot shows a web browser window with the URL 'localhost:3000/createGroup'. The page title is 'React App'. The navigation bar includes links for 'Home', 'Banko!', 'Join Group', 'Create Group', 'Create Transaction', and 'Log out'. The main content area is titled 'Create A New Group' and contains a single input field labeled 'Team Banko' and a 'Submit' button. At the bottom of the page, there are links for 'About Us!', 'Banko!', and social media profiles for 'Gabe Github', 'Ajit Github', and 'Osama Github'.

The new group is now added in the database.

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree, with the 'omjmf6vzmpqpgc0p' schema selected. Under this schema, the 'Tables' node is expanded, showing tables like address, bill, city, country, group\_list, message, message\_content, payment, transaction, user, and user\_in\_group. Below these are 'Views', 'Stored Procedures', and 'Functions'. The bottom-left pane shows 'Administration' and 'Information' tabs, with 'No object selected' currently active. The main workspace is titled 'group\_list' and contains a SQL editor with the query 'SELECT \* FROM omjmf6vzmpqpgc0p.group\_'. Below the query is a 'Result Grid' showing the contents of the 'group\_list' table. The table has three columns: 'group\_id', 'group\_name', and 'date\_created'. The data includes 19 rows, starting with group\_id 4 and ending with group\_id 19. The 'group\_list1' tab is visible at the bottom of the grid.

group_id	group_name	date_created
4	The Three Musketeers	2020-10-31 00:16:41
5	Fantastic 4	2020-10-31 00:17:51
6	Dream Team	2020-10-31 00:18:13
7	Full House	2020-10-31 00:19:28
8	Pearfect Roommates	2020-10-31 00:20:02
9	The Fab Five	2020-10-31 00:20:32
10	Game of Phones	2020-10-31 00:20:52
11	The Trouble Makers	2020-10-31 00:22:52
12	Best Buddies	2020-10-31 00:23:48
13	The Dreamers	2020-10-31 00:24:16
14	Happy Home	2020-10-31 00:26:33
15	Family	2020-10-31 00:27:59
16	Team 7	2020-12-01 00:40:42
17	The best coders	2020-12-01 05:33:17
18	CS157A	2020-12-02 01:38:28
19	Team Banko	2020-12-11 03:08:00

We check to see if the group is unique. If it is, we go on to try to create the group. If the statement passes, then we have the user join the group that they made, and then return with a successful flag.

```
public int createBankGroup() throws SQLException {

    int groupCreated = 0;
    Connection connection = BancoBackendServer.connection;
    if (uniqueGroup(group_name, connection)) {
        // Group is created
        if (createGroup(group_name, connection)) {

            if (userJoinGroup()) {
                // User join the group created
                groupCreated = 1;
            }
        }
    } else {
        // Group name is not unique
        groupCreated = -1;
    }

    return groupCreated;
}
```

```
private boolean createGroup(String group_name, Connection connection) {
    boolean flag = false;

    String query = "INSERT INTO omjmf6vzmpqpgc0p.group_list (group_name, date_created) VALUES ('" + group_name + "', now())";

    try {
        Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                                       ResultSet.CONCUR_UPDATABLE);
        statement.executeUpdate(query);
        flag = true;
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}

return flag;
```

## Join Group Functionality:

Here is the frontend for the join group page. We can navigate to the page through the navbar by clicking on the join group tab.

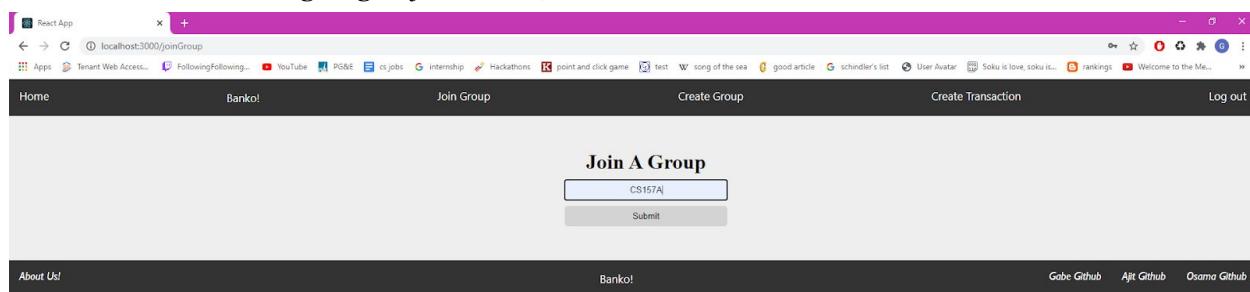
The screenshot shows a web browser window with a pink header bar. The address bar displays 'localhost:3000/joinGroup'. The main content area has a title 'Join A Group' and two input fields: 'Group Name' and 'Submit'. Below the form is a footer with links: 'About Us!', 'Banko!', 'Gabe Github', 'Ajit Github', and 'Osama Github'. The top navigation bar includes links for 'Home', 'Banko!', 'Join Group', 'Create Group', 'Create Transaction', and 'Log out'.

We can join an existing group called 'CS157A'. The group 'CS157A' has the group\_id = 18 and the user MikuWu has the user\_id = 37. Here is the user\_in\_group table in our database before we join a group.

The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane shows the schema 'omjmf6vzmpqpgc0p' with its tables: address, bill, city, country, group\_list, message, message\_content, payment, transaction, user, and user\_in\_group. The 'user\_in\_group' table is selected. The SQL Editor pane at the top contains the query: 'SELECT \* FROM omjmf6vzmpqpgc0p.user\_in\_'. The Results pane below shows the data for the user\_in\_group table:

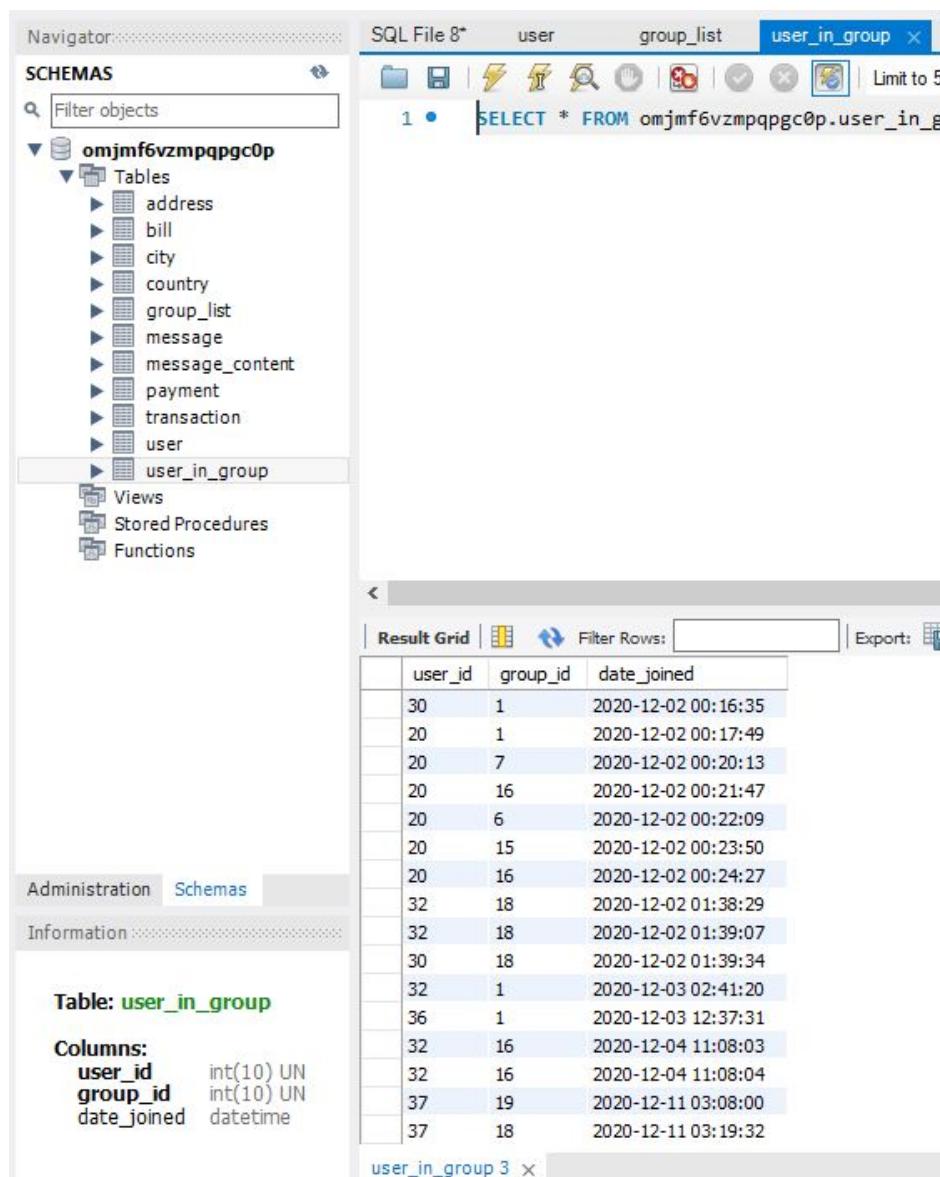
user_id	group_id	date_joined
20	7	2020-12-01 22:55:21
30	1	2020-12-02 00:16:35
20	1	2020-12-02 00:17:49
20	7	2020-12-02 00:20:13
20	16	2020-12-02 00:21:47
20	6	2020-12-02 00:22:09
20	15	2020-12-02 00:23:50
20	16	2020-12-02 00:24:27
32	18	2020-12-02 01:38:29
32	18	2020-12-02 01:39:07
30	18	2020-12-02 01:39:34
32	1	2020-12-03 02:41:20
36	1	2020-12-03 12:37:31
32	16	2020-12-04 11:08:03
32	16	2020-12-04 11:08:04
37	19	2020-12-11 03:08:00

In the frontend we are going to join 'CS157A'.



The screenshot shows a web application interface. At the top, there's a navigation bar with links like 'Home', 'Banko!', 'Join Group', 'Create Group', 'Create Transaction', and 'Log out'. Below the navigation bar, the main content area has a title 'Join A Group'. In the center, there's an input field containing 'CS157A|' and a 'Submit' button below it. At the bottom of the page, there's a footer with links for 'About Us!', 'Banko!', and three GitHub profiles: 'Gabe GitHub', 'Ajit GitHub', and 'Osama GitHub'.

Here is the database for users\_in\_group. Now you can see that group\_id = 18 and user\_id = 37 are now inserted into the table.



The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane displays the schema 'omjmf6vzmpqpgc0p' with its tables: address, bill, city, country, group\_list, message, message\_content, payment, transaction, user, and user\_in\_group. The 'user\_in\_group' table is selected. The SQL tab shows a query: `SELECT * FROM omjmf6vzmpqpgc0p.user_in_g`. The Results tab displays the data in a grid:

	user_id	group_id	date_joined
30	1	2020-12-02 00:16:35	
20	1	2020-12-02 00:17:49	
20	7	2020-12-02 00:20:13	
20	16	2020-12-02 00:21:47	
20	6	2020-12-02 00:22:09	
20	15	2020-12-02 00:23:50	
20	16	2020-12-02 00:24:27	
32	18	2020-12-02 01:38:29	
32	18	2020-12-02 01:39:07	
30	18	2020-12-02 01:39:34	
32	1	2020-12-03 02:41:20	
36	1	2020-12-03 12:37:31	
32	16	2020-12-04 11:08:03	
32	16	2020-12-04 11:08:04	
37	19	2020-12-11 03:08:00	
37	18	2020-12-11 03:19:32	

User joins a group by sending a POST with the unique group name and if they are not in the group already, it enters them into the group.

```
public boolean userJoinGroup() throws SQLException {
    Connection connection = BancoBackendServer.connection;
    group_id = getGroupId(group_name, connection);
    String query = "INSERT INTO omjmft6vzmpqpgc0p.user_in_group (user_id, group_id, date_joined) VALUES (" + this.user_id + "," +
        + this.group_id + ", now())";

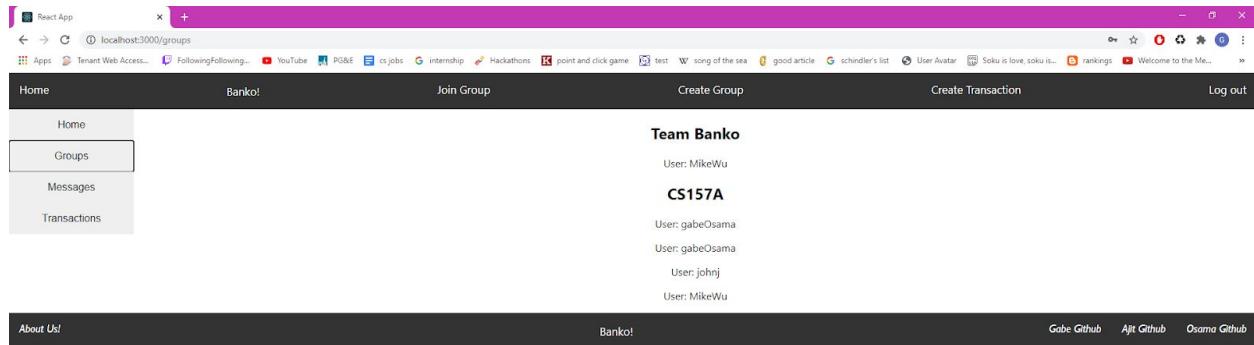
    try {
        Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
        statement.executeUpdate(query);

        return true;
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }

    return false;
}
```

## The Groups Page Functionality:

In the frontend there is a sidebar to navigate to the groups page. In this page it shows the current groups that the user is currently in. Here it shows that we are in the groups Team Banko and CS157A. We create Team Banko and thus we are the only user in the group, but there are multiple members in CS157A.



Queries the backend with a GET request, asking for the group information for the user currently logged in. The backend fetches the data and returns it here. We then map each group to a div.

```
function getGroups() {
    $.ajax({
        contentType: "application/json; charset=utf-8",
        url: 'http://localhost:8080/userGroups',
        type: 'get',
        dataType: 'json',
        data: { user_id: state.id },
        success: function (data) {

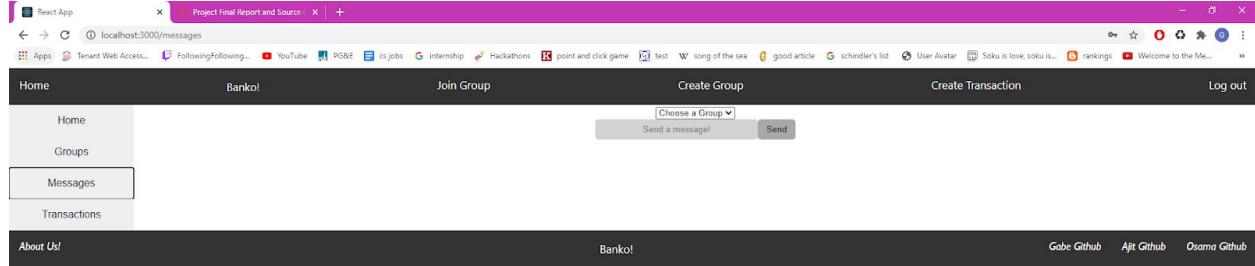
            var i = 0;
            setGroups(data);
        },
        error: function (request, status, error) {
            console.log(request.responseText);
        }
    });
}

if (flag == 0) {
    getGroups();
    setFlag(1);
}

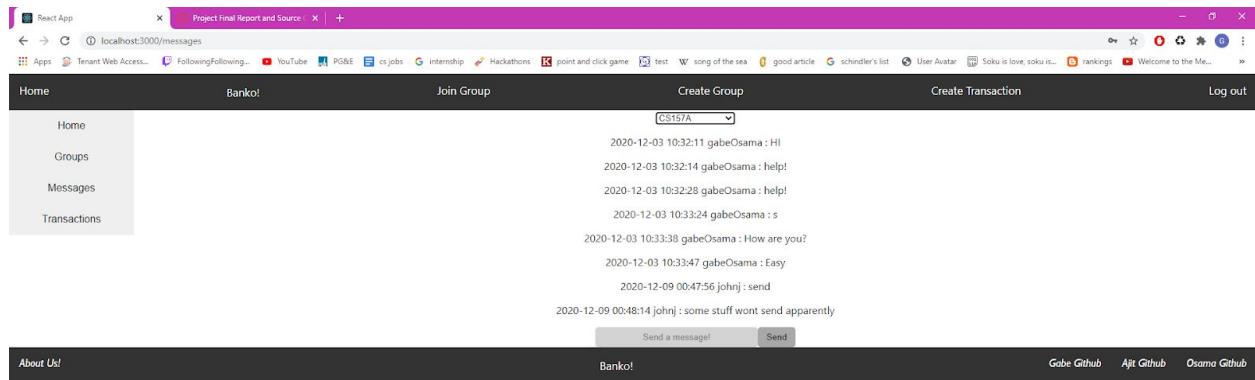
if (flag == 1) {
    return (
        <div className={styles.GroupsFeed}>
            <div className={styles.Groups}>
                {groups.map(group => (
                    <Group group_id={group.group_id} group_name={group.group_name} />
                )));
            </div>
        </div>
    )
}
```

## Messages Page and Functionality:

In the frontend there is a sidebar to navigate to the Messaging page. In the Messaging page you will see a dropdown box with the current groups you are in.



When you click on a specified group in the dropdown box, you will enter the group chat of that group. The Messaging page will indicate which user has sent which message and the time date.



We send a request to get the messages. The backend then gets all the content using a triple join to put all the data that we need together, and a subquery to get the groupid. We then order by the date sent. This data is then stored in an arraylist of hashmaps, which acts as a json object when returning it to the front end.

```
public static ArrayList<HashMap<String, String>> getUserGroupMessage(String group_name) throws SQLException {
    Connection connection = BankoBackendServer.connection;
    ArrayList<HashMap<String, String>> listGroupMessages = new ArrayList<HashMap<String, String>>();

    // nested query for getting group id
    String getGroupID = "(SELECT group_id FROM omjmf6vzmpqpgc0p.group_list WHERE group_name = '" + group_name
        + "')";

    String selectSql = "SELECT message_content_id, content, group_id, user_id, username, date_sent FROM omjmf6vzmpqpgc0p.message "
        + "JOIN omjmf6vzmpqpgc0p.message_content USING (message_content_id) JOIN omjmf6vzmpqpgc0p.user USING (user_id) WHERE (group_id = "
        + getGroupID + ") ORDER BY date_sent ASC";
    Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
    ResultSet rs = statement.executeQuery(selectSql);
    while (rs.next()) {
        HashMap<String, String> userGroupMessageJSON = new HashMap<String, String>();
        userGroupMessageJSON.put("message_content_id", Integer.toString(rs.getInt("message_content_id")));
        userGroupMessageJSON.put("content", rs.getString("content"));
        userGroupMessageJSON.put("group_id", Integer.toString(rs.getInt("group_id")));
        userGroupMessageJSON.put("user_id", Integer.toString(rs.getInt("user_id")));
        userGroupMessageJSON.put("username", rs.getString("username"));
        userGroupMessageJSON.put("date_sent", (rs.getString("date_sent")));
        listGroupMessages.add(userGroupMessageJSON);
    }
    statement.close();
    return listGroupMessages;
}
```

Now in this same page we have the functionality of sending a new message to the group. There is a text box to send a message. In this case we are going to send “I am the professor of CS157A” to the group chat.

The screenshot shows a web application interface for a group chat. The top navigation bar includes links for 'React App', 'Project Final Report and Source', and a search bar. Below the navigation is a header with tabs for 'Home', 'Banko!', 'Join Group', 'Create Group', 'Create Transaction', and 'Log out'. The 'Groups' tab is currently active. On the left, a sidebar menu has 'Groups' selected. The main content area displays a list of messages:

- 2020-12-03 10:32:11 gabeOsama : Hi
- 2020-12-03 10:32:14 gabeOsama : help!
- 2020-12-03 10:32:28 gabeOsama : help!
- 2020-12-03 10:33:24 gabeOsama : s
- 2020-12-03 10:33:38 gabeOsama : How are you?
- 2020-12-03 10:33:47 gabeOsama : Easy
- 2020-12-09 00:47:56 johnj : send
- 2020-12-09 00:48:14 johnj : some stuff wont send apparently

A text input field at the bottom contains the message 'I am the professor of CS157A' and a 'Send' button. The footer features links for 'About Us!', 'Banko!', and three GitHub profiles: 'Gabe Github', 'Ajit Github', and 'Osama Github'.

Here is the database before we send an Ajax request to the backend. Here is the message\_content table.

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the schema `omjmf6vzmpqpgc0p` with tables: address, bill, city, country, group\_list, message, and message\_content.
- SQL Editor:** A query window with the SQL command:
 

```
1 •  SELECT * FROM omjmf6vzmpqpgc0p.message;
```
- Result Grid:** Displays the data from the message\_content table:
 

message_content_id	group_id	user_id
46	1	32
47	1	32
48	1	32
49	1	32
50	1	32
51	1	36
52	1	36
53	1	30
54	17	29
55	1	32
56	1	32
57	1	32
58	1	32
59	1	32
60	18	30
61	18	30
- Table Information:**
  - Table: message\_content**
  - Columns:**

message_content_id	content
int(11)	content
UN	date
PK	content
varchar	date

Here is the frontend after we send the message. It now appears in the chat box.

The screenshot shows a web browser window with the following details:

- Header:** React App - Project Final Report and Source
- Address Bar:** localhost:3000/messages
- Navigation:** Home, Banko!, Join Group, Create Group, Create Transaction, Log out
- Left Sidebar:** Home, Groups, Messages, Transactions
- Right Content Area:**
  - A dropdown menu shows "CS157A".
  - Message history:
    - 2020-12-03 10:32:11 gabeOsama : HI
    - 2020-12-03 10:32:14 gabeOsama : help!
    - 2020-12-03 10:32:28 gabeOsama : help!
    - 2020-12-03 10:33:24 gabeOsama : s
    - 2020-12-03 10:33:38 gabeOsama : How are you?
    - 2020-12-03 10:33:47 gabeOsama : Easy
    - 2020-12-09 00:47:56 johnj : send
    - 2020-12-09 00:48:14 johnj : some stuff wont send apparently
    - 2020-12-11 03:44:01 MikeWu : I am the professor of CS157A
  - Input fields: "Send a message!" and "Send"
  - Footer links: About Us!, Banko!, Gabe Github, Ajit Github, Osama Github

Here is the database after the new message is sent. The group\_id for CS157A is 18, the user\_id is 37. A new row has been inserted with a new message\_content\_id.

The screenshot shows the MySQL Workbench interface. The left pane displays the Navigator with the schema 'omjmf6vzmpqpgc0p' selected. Under 'Tables', the 'message\_content' table is expanded, showing its columns: message\_content\_id, group\_id, and user\_id. The right pane shows the results of the SQL query 'SELECT \* FROM omjmf6vzmpqpgc0p.message;'. The result grid contains 62 rows of data:

message_content_id	group_id	user_id
47	1	32
48	1	32
49	1	32
50	1	32
51	1	36
52	1	36
53	1	30
54	17	29
55	1	32
56	1	32
57	1	32
58	1	32
59	1	32
60	18	30
61	18	30
62	18	37

Here is the database for the message\_content table. Here you can see the message\_content\_id = 62 is now inserted with the message content of “I am the professor of CS157A”.

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** The current schema is `omjmf6vzmpqpgc0p`.
- Tables:** The `message_content` table is selected.
- SQL Editor:** The query `SELECT * FROM omjmf6vzmpqpgc0p.message_content;` is displayed.
- Result Grid:** The table structure and data are shown in a grid format.

	message_content_id	content	date_sent
48		Whats up guys?	2020-12-03 12:11:46
49		How are you doing?	2020-12-03 12:11:52
50		I am doing pretty well over here!	2020-12-03 12:11:57
51		Hello	2020-12-03 12:39:29
52		pizza	2020-12-03 12:39:53
53		Hey!	2020-12-04 00:39:15
54		We the best	2020-12-04 02:00:03
55		Hi Sam!	2020-12-04 11:05:31
56		<h1>hi</h1>	2020-12-04 11:05:44
57		"c SELECT * FROM users;	2020-12-04 11:06:10
58		c	2020-12-04 11:06:51
59		"\c	2020-12-04 11:07:37
60		send	2020-12-09 00:47:56
61		some stuff wont send apparently	2020-12-09 00:48:14
62	*	I am the professor of CS157A	2020-12-11 03:44:01

We have the addMessageContent functionality which adds the message content to a content table, and then we add the id for the message content to the message table using addMessage.

```

private static int addMessageContent(String content, Connection connection) {
    String query;

    query = "INSERT INTO omjmf6vzmpqpgc0p.message_content (content, date_sent) VALUES ('" + content + "', NOW())";

    try {
        Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_UPDATABLE);

        statement.executeUpdate(query);

        query = "SELECT message_content_id FROM omjmf6vzmpqpgc0p.message_content ORDER BY message_content_id DESC LIMIT 1";

        ResultSet rs = statement.executeQuery(query);
        // gets last inserted key, aka the content id we just inserted into the table
        if (rs.next())
            return rs.getInt("message_content_id"); // returns the message content id

    } catch (SQLException e) {
        e.printStackTrace();
    }
    return -1;
}

private static int addMessage(int messageContentID, String group_name, int user_id, Connection connection) {
    String query;
    String groupid = "(SELECT group_id FROM omjmf6vzmpqpgc0p.group_list WHERE group_name = '" + group_name + "')";

    query = "INSERT INTO omjmf6vzmpqpgc0p.message (message_content_id, group_id, user_id) VALUES (
        " + messageContentID + ", " + groupid + ", " + user_id + ")";

    try {
        Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_UPDATABLE);

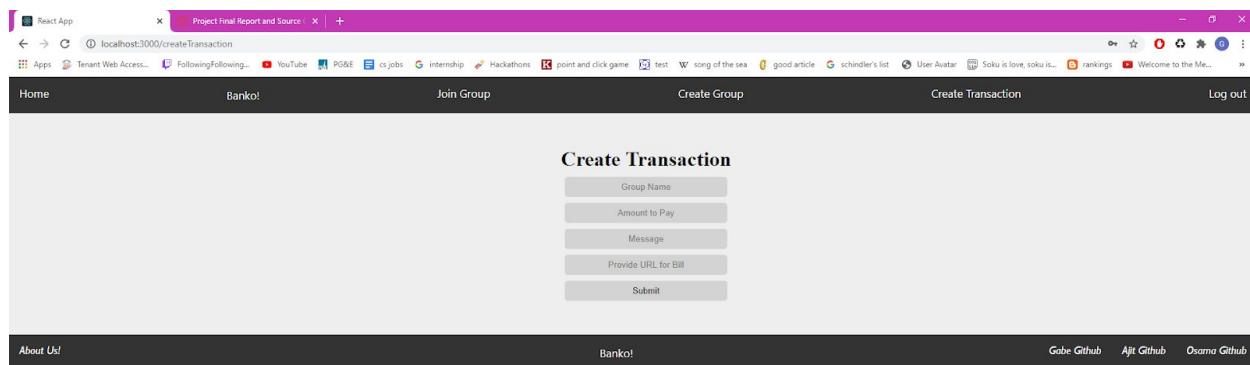
        statement.executeUpdate(query);

        return 1;
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return -1;
}

```

## Create Transaction Functionality:

Here is the frontend for our create transaction page. We can navigate to this page by clicking the Create Transaction tab on the navbar on the top.



Here is the Transactions table in the database before we insert a new transaction from the frontend.

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the schema `omjmf6vzmpqpgc0p` containing tables like address, bill, city, country, group\_list, message, message\_content, payment, transaction, user, and user\_in\_group.
- SQL File 8\***: A query window with the command `SELECT * FROM omjmf6vzmpqpgc0p.transaction;`
- Result Grid:** Displays the data from the transaction table.
- Table Structure:** Shows the columns and data types for the transaction table.

**Table: transaction**

**Columns:**

	transaction_id	group_id	date_created	date_closed	transaction_content	amount
18	1		2020-12-02 01:28:28	2020-12-04 00:39:40	rent	40.00
19	3		2020-12-02 01:28:52	HULL	rent	40.00
20	18		2020-12-09 01:19:28	2020-12-09 01:37:57	Pizza	40.00
21	16		2020-12-09 01:20:05	HULL	Pizza	40.00
22	1		2020-12-09 01:22:44	2020-12-09 01:38:06	Pizza	40.00
23	18		2020-12-09 01:32:46	2020-12-09 01:38:08	null	30.00
24	18		2020-12-09 01:37:32	HULL	Pay up Osama	50.00
25	18		2020-12-09 02:08:47	HULL	Rent	60.00
26	18		2020-12-09 03:00:49	HULL	hello test!	500.00
27	18		2020-12-09 03:02:32	HULL	5	5.00
28	18		2020-12-09 03:02:37	HULL	5	5.00
29	16		2020-12-09 03:36:48	HULL	testing image for bill	40.00
30	16		2020-12-09 03:42:39	HULL	another test	50.00
31	18		2020-12-11 01:46:13	HULL	Pay up Osama	50.00
32	16		2020-12-11 01:56:08	HULL	Doggy	4000.00

Here is the frontend when you create a new transaction. We are indicating for which group and how much each person pays along with a description of the transaction. We can also provide a url link to the bill and it will store in the bills table.

The screenshot shows a web application window titled 'React App'. The URL is 'localhost:3000/createTransaction'. The page has a header with 'Home', 'Banko!', 'Join Group', 'Create Group', 'Create Transaction', and 'Log out'. Below the header is a form titled 'Create Transaction' with fields for 'group' (set to 'CS157A'), 'amount' (set to '1000'), and 'description' (set to 'Rent'). A 'Submit' button is present. Below the form, a message 'Bill Added' is displayed. At the bottom of the page, there are links for 'About Us!', 'Banko!', 'Gabe GitHub', 'Ajit GitHub', and 'Osama GitHub'.

Here is the transactions table in the database after we created the new transaction. A new row has been created with the transaction\_content, group\_id and amount.

The screenshot shows a PostgreSQL database client interface. On the left, the 'SCHEMAS' tree view shows the 'omjmf6vzmpqpgc0p' schema with tables like address, bill, city, country, group\_list, message, message\_content, payment, transaction, user, and user\_in\_group. The 'Views', 'Stored Procedures', and 'Functions' sections are also visible. In the center, a query editor window displays the SQL command: 'SELECT \* FROM omjmf6vzmpqpgc0p.transaction;'. Below the query is a 'Result Grid' table with the following data:

	transaction_id	group_id	date_created	date_closed	transaction_content	amount
19	3	2020-12-02 01:28:52	NULL	rent	40.00	
20	18	2020-12-09 01:19:28	2020-12-09 01:37:57	Pizza	40.00	
21	16	2020-12-09 01:20:05	NULL	Pizza	40.00	
22	1	2020-12-09 01:22:44	2020-12-09 01:38:06	Pizza	40.00	
23	18	2020-12-09 01:32:46	2020-12-09 01:38:08	null	30.00	
24	18	2020-12-09 01:37:32	NULL	Pay up Osama	50.00	
25	18	2020-12-09 02:08:47	NULL	Rent	60.00	
26	18	2020-12-09 03:00:49	NULL	hello test!	500.00	
27	18	2020-12-09 03:02:32	NULL	5	5.00	
28	18	2020-12-09 03:02:37	NULL	5	5.00	
29	16	2020-12-09 03:36:48	NULL	testing image for bill	40.00	
30	16	2020-12-09 03:42:39	NULL	another test	50.00	
31	18	2020-12-11 01:46:13	NULL	Pay up Osama	50.00	
32	16	2020-12-11 01:56:08	NULL	Doggy	4000.00	
33	18	2020-12-11 04:08:09	NULL	Rent	1000.00	

The bills table has also been updated with the url link for each transaction.

The screenshot shows the MySQL Workbench interface. On the left, the Schemas tree displays the 'omjmf6vzmpqpgc0p' schema with various tables like address, bill, city, country, group\_list, message, message\_content, payment, transaction, user, and user\_in\_group. The 'message\_content' table is selected. Below the schema tree, the 'Administration' tab is active, followed by 'Schemas' and 'Information'. The main area shows the 'SQL File 8\*' tab with the query 'SELECT \* FROM omjmf6vzmpqpgc0p.bill;'. To the right is the 'Result Grid' showing the data from the 'message\_content' table:

	bill_id	transaction_id	photo_url
17	1		www.google.com/photoOfOsamaSmiling
18	1		www.google.com/photoOfOsamaSmiling
19	1		www.google.com/photoOfOsamaSmiling
20	1		www.google.com/photoOfOsamaSmiling
21	1		www.google.com/photoOfOsamaSmiling
22	1		www.google.com/photoOfOsamaSmiling
23	24		www.test.com
24	25		www.renpicture.com
25	26		https://github.com/osamahan999
26	27		5
27	28		5
28	29		https://en.wikipedia.org/wiki/Image#/media/Fil...
29	30		https://www.talkwalker.com/images/2020/blog-...
30	32		https://www.planetware.com/wpimages/2020/...
31	31		https://www.pge.com/pge_global/local/images/...
*	NULL	NULL	NULL

Takes in the transaction details, and makes the transaction itself.

```
public int createTransaction() throws SQLException {
    Connection connection = BancoBackendServer.connection;
    group_id = getGroupId(group_name, connection);
    String query = "INSERT INTO omjmf6vzmpqpgc0p.transaction (group_id, date_created, transaction_content, amount) VALUES(" +
        + group_id + ", NOW(), '" + transaction_content + "', " + amount + ")";
    try {
        Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
        statement.executeUpdate(query);
        return 1;
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return -1;
}
```

When creating a transaction, it sends back the transaction id and we add the bill to that transaction.

```
/** 
 * Adds a bill which is just a photo linked to a certain transaction returns 1
 * for succes, 0 for error
 *
 * @return
 */
public int createBill() throws SQLException {
    Connection connection = BancoBackendServer.connection;
    group_id = getGroupId(group_name, connection);
    transaction_id = getTransactionId(group_id, connection);

    String query = "INSERT INTO omjmf6vzmpqpgc0p.bill (transaction_id, photo_url) VALUES (" + transaction_id + ", '" +
        + photo_url + "')";
    try {
        Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
        statement.executeUpdate(query);
        return 1;
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return -1;
}
```

## Transaction Page Functionality:

Here is the frontend for the transaction page. To navigate to the transaction page is by clicking the transaction tab on the sidebar on the left. In this page it shows the current transactions for each group you are in. It also shows the new transaction we just made previously.

The screenshot shows a web browser window titled "React App" with the URL "localhost:3000/transactions". The page displays a list of transactions in a table format. Each row represents a transaction with the following columns: amount (\$), message, timestamp, and a "Show Bill" button. The last transaction in the list is highlighted with a red border and contains a "Pay Now" button. The footer of the page includes links for "About Us!", "Banko!", and GitHub profiles for Gabe, Ajit, and Osama.

\$500.0	This is <b>paid</b> on 2020-12-11 04:14:07	<a href="#">Show Bill</a>
\$5.0	Message: 5 \$5.0	
\$5.0	This is <b>paid</b> on 2020-12-11 04:14:32	<a href="#">Show Bill</a>
\$5.0	Message: 5 \$5.0	
\$5.0	This is <b>paid</b> on 2020-12-11 04:14:31	<a href="#">Show Bill</a>
\$50.0	Message: Pay up Osama \$50.0	
\$50.0	This is <b>paid</b> on 2020-12-11 04:14:30	<a href="#">Show Bill</a>
\$1000.0	Message: Rent \$1000.0	
\$1000.0	This is <b>unpaid</b>	<a href="#">Pay Now</a>

In the frontend, we can pay each transaction by pressing the Pay Now button. After clicking the button it now says the transaction has been paid!

The screenshot shows the same transaction page after the "Pay Now" button was clicked for the last transaction. The transaction row now has a green background color and displays the message "This is **paid**". The footer of the page includes links for "About Us!", "Banko!", and GitHub profiles for Gabe, Ajit, and Osama.

In the frontend, we can also show or hide a bill for a specific transaction.

---

Message: Doggy \$4000.0

\$4000.0

This is  
unpaid:

[Pay Now](#)

[Show Bill](#)

We can click Show Bill, which renders the image from the database.

Message: Doggy \$4000.0

\$4000.0

This is  
unpaid:

[Pay Now](#)

[Show Bill](#)



We do a full outer join between transaction and bill to get all the transaction data with nulls for where we do not have bills. This allows the parsing in the front end to work much more smoothly.

```

    for (HashMap<String, String> map : listTransactions) {
        for (String s : map.keySet()) {
            String group_id = map.get(s);
            int tempGroup_id = Integer.parseInt(group_id);
            String query = "SELECT transaction.transaction_id, group_id, date_created, date_closed, transaction_content, "
                + " amount, photo_url FROM omjmf6zmpqpc0p.transaction LEFT JOIN omjmf6zmpqpc0p.bill "
                + " ON transaction.transaction_id = bill.transaction_id WHERE transaction.group_id ="
                + tempGroup_id
                + " UNION SELECT transaction.transaction_id, group_id, date_created, date_closed, transaction_content, amount, photo_url "
                + " FROM omjmf6zmpqpc0p.transaction RIGHT JOIN omjmf6zmpqpc0p.bill ON transaction.transaction_id = "
                + " bill.transaction_id WHERE transaction.transaction_id IS NULL";
        try {
            Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_UPDATABLE);

            ResultSet rs = statement.executeQuery(query);
            while (rs.next()) {
                HashMap<String, String> userAllTransactionJSON = new HashMap<String, String>();
                userAllTransactionJSON.put("transaction_id", Integer.toString(rs.getInt("transaction_id")));
                userAllTransactionJSON.put("group_id", Integer.toString(rs.getInt("group_id")));
                userAllTransactionJSON.put("date_created", (rs.getString("date_created")));
                userAllTransactionJSON.put("date_closed", (rs.getString("date_closed")));
                userAllTransactionJSON.put("transaction_content", (rs.getString("transaction_content")));
                userAllTransactionJSON.put("amount", Double.toString(rs.getDouble("amount")));
                userAllTransactionJSON.put("photo_url", (rs.getString("photo_url")));

                listAllGroupTransactions.add(userAllTransactionJSON);
            }
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
    }
}

```

The code for showing the bills is quite simple; we use states to show or not show the bill, and change the state based on the button press.

```

<p>{props.transaction_content} ${props.amount}</p>
{props.date_closed == "0000-00-00 00:00:00" || props.date_closed == null} ?
    <div className={styles.Transaction}-This is <b>unpaid:</b> <form onSubmit={handleSubmit(payTransaction)}>
        <button name="event" value={jsonData} ref={register({ required: true })} > Pay Now</button>
    </form></div> : <div>This is <b>paid</b> on {props.date_closed}
</div>

```

### **Generating Visuals (Pie charts and Graphs):**

This functionality was not implemented. This was not done due to lack of time. We found a way to do this using react, but none of us had the knowledge to try to write it initially. This seemed like a specialty feature, and so we pushed it off for more important functionalities.

### **Splitting the bill Functionality:**

This functionality was not implemented. This was not finished simply due to lack of time. This would be a bit complicated to implement as we would need to store how users are paying a certain transaction. Perhaps this could be done using a new table request-for-payment which holds group #, user #, transaction #, and the type of payment. For now though, this is left for the user to do outside of our application.

### **Store Receipts after payment Functionality:**

This functionality was not implemented because we did not have enough time as we struggled with connecting the front-end with back-end. But we would work on it for future improvements, we would implement this by storing the user's proof of payment image or the url for the image in the database when the user makes a payment for a transaction.

### **User Permissions In Groups:**

The functionalities for users being able to accept other users or remove other users from a group was not implemented. We simply did not have time, but the implementation would not be very difficult. We would need a table that holds the user permissions, and then a table that holds join requests. If a user tries to accept another user, we would simply need to check whether they have the permissions for that group, and if they do, we put the user into the group. Same thing with removal, except the removal request is sent by the user with the permissions.

### **Tag bills with title, keywords, and date to be able to search them and filter:**

The functionality was not implemented but we would have implemented it by doing a MySQL query and having the search keyword as the condition in the where clause of the query and filter would have been done by selecting specific conditions in the date\_created and date\_closed. Can also search the content of the bills using a wildcard and whatever the user inputs.

### **Approval system for payments:**

This was not implemented due to a lack of time. How we would have implemented this though, is add an intermediary table between a user paying a transaction, and the transaction being marked as paid. It puts in a request to pay, and sends to the group owner (based on the permissions we had not implemented) that someone is trying to pay. The group owner then either says "ok" and the request goes through, or the owner says "no" and it is denied.

### **Optional Functionalities:**

We simply did not have the time to implement the venmo requests, the email confirmations, nor the profile pictures.

## Procedure (Step by Step) of How to Set up and Run Website

- **To run this, you must first clone,**
  - <https://github.com/osamahan999/CS157A-section9-team6.git> repo.
  - **Make sure you have NodeJS, NPM and Java version 8**
- **Front-End Dependencies**
  - **cd into folder '/banko'**
  - **rm node\_modules -r**
  - **npm install**
  - **npm install jquery**
  - **npm install react-hook-form**
  - **npm install react-router-dom**
- **Run system**
  - **Open the file in 'banko-backend/BankoBackendServer'**
  - **Run the main method (Backend Server Starts)**
  - **Go into the 'banko' folder**
  - **Run “npm start”**
  - **In your browser go to url: http://localhost:3000**

## Lessons Learned

Osama :

I learned a lot throughout this project due to the vast technologies we used. It was my first time using Spring Boot, and React Js. Learning to use hooks and controllers and ajax to send requests was both frustrating and exciting. We used postman to make sure our backend worked, and that was also interesting as we learned about HTTP requests and the different kinds of them. I also learned how to write joins, and had a lot of fun trying to figure out a way to write the full outer join for our transaction functionalities.

Gabriel :

For this project I learned how to use multiple technologies to create an application that stores and gets information to a database. It was my first time doing a project of this scale and my first time using Java Spring for the backend and React Js for the frontend. I learned how requests are made using Ajax requests from React to the Java Spring backend. I learned how to manipulate the contents given from each POST and GET requests. I also learned how to use the application Postman to send requests to the backend to make sure that our backend is running correctly. Finally I learned how to set up the different ports for React: 3000 and be able to connect it to the Tomcat server through port 8080. This project was hard, yet it was a great experience.

Ajit :

In this course project, I learnt about setting up 3-tier architecture and how MySQL database interacts with the backend. In this project we used Java Spring which none of us (team members) had prior knowledge about but there were many websites and videos which helped us create the Java Spring project. Our main reason for using Java Spring was for using rest apis to connect data from front-end and back-end. Making GET and POST methods for each java function class. One problem which we encountered was we were running the react.js (front-end) on the same port as Tomcat Server and after some research we found out we had to cross-origin and have them running on different ports. React.js: port 3000 and Tomcat Server: port 8080. Overall I applied my MySQL query knowledge into writing queries in the backend.