Stonk: Stock Trading Website

Team 1

Ryota Suzuki, Justin Leung, Sawyer Larsh, Xinbao huang

Introduction

The Stonks stock platform will allow users to interact with stocks. This platform will allow users to trade stocks as well as monitor stocks. Users can trade based on the stocks current value as well as transfer their funds to and from a bank account. Transaction between users will be a manual 1 for 1. Meaning, users will need to post a request with their desired quantity and price, and other users must accept the request for a transaction to be successful. Once successful, then the database will update shares owned, and funds between the two users.

The purpose of this application is for users around the globe to buy and sell stocks as well as monitor the current value of stock prices. Users can check the prices of stocks as well as buy stocks and sell stocks based on their desired value.

Operations for Actors

In our stock trading app, we have general users as well as admins. Users are allowed to post a sales or buy request, and also accept other user's sales and buy requests. The admins are able to do all user operations, as well as deleting sales and buy requests if it is inappropriate, fishy, or other reasons. The operations for each actor are as follows.

User

● Create Request sale	**Done**
● View active sale request	**Done**
● Buy from sale request	**Done**
● View general info of stocks	**Done**
● Reset password	**Done**
● Add funds and bank account	**Done**
● Transfer funds to bank account	**Not implemented; elaborated on in post mortem**

Admin

● All user operations	**Done**
● Delete sale request	**Done**
● Delete shares	**Done**

Scenarios

A user named Ryan was interested in investing in stocks so he decided to start it. He found a investment website called, “Stonks” and registers with his email and creates a password. Then, he looked at the various stocks and deposited $10000 to the account to buy stocks. He decided to buy a couple from Paypal, Microsoft, and Tesla. After buying, he logged out and left it unwatched for 2 weeks. He decided to hop back on “Stonks” and sees that Microsoft and Paypal are on a uphill rise but Tesla stocks going straight down. Ryan decided to sell the Tesla stock at the market value and decided that he will invest more into Paypal. He then logs out and comes back in again after 2 weeks. Ryan, after looking at Paypal and Microsoft going down in stock value, he decided that it was enough for him. He traded in all his stocks and withdrew the money in his account into his bank account (attribute).

Members Contribution

Ryota

- Defined data requirements, functionalities, and operations
- Created, and revised proposal
- Created, and revised ERD
- Created database schema
- Created database using MySQL
- Created sample data for database
- Created mock user page layout and structure with pseudo queries
- Wrote project term report

Justin

- Assisted on writing of proposal
- Worked on ERD

- Set up MySql Instance on AWS RDS
- Added sample data for database
- Created admin page layout and structure with necessary queries
- Created admin pages on application
- Reformatted SQL tables to fit our requirements for the application
- Also worked on the report

Sawyer

- Revised data requirements, functionalities, and operations
- Contributed to proposal
- contributed to ERD
- Created EC2 machine, installed all necessary applications (apache2) and dependencies including php and MySQL connector
- Configured AWS to allow connections to the App server for ssh (team members only) and http (global) as well as access to RDS
- Revised schema to reflect requirements
- Revised MySQL database to support application
- Created sample data for database
- Created register page, login page, user page as well as all pages referenced by user pages including writing all HTML and PHP code and designing/revising queries used in PHP for the application
- Outlined and performed testing of application
- Wrote project term report

Xinbao

- Created account for AWS RDS
- Assisted testing stored procedures on MySQL

Project Choice

For our term project, we decided to create a database for a stock trading website called Stonk. Stonk is a stock trading platform that allows users to buy, sell, and own stocks. It is a market place, meaning users can create their own sales and set the pricing and quantity themselves.

Application Design

Our application is a web application using a LAMP stack. We use Amazon's Relational DataBase Server (MySQL based) as our database. This is a serverless application on AWS so we can focus on the database itself, and all of the VM related portions like load balancing and redundancy are orchestrated by AWS. Our application uses PHP with the PHP MySQL connector. This is run on the Amazon Linux AMI operating system on an EC2 virtual machine in AWS. We also used the Bootstrap 4 framework, which uses HTML and CSS. This enabled the visual formatting without creating CSS style sheets and designing the look ourselves. Other tools and languages used for development include MySQL workbench, OpenSSH, Bash, SCP, GitHub.

The following languages are used in the project:

- HTML
- PHP
- CSS

Data Requirements

- The platform is composed of many stock accounts. Each of them has a unique user_id and unique password which is hashed, and unique email. Users can only have one account tied to their email. Users must deposit money into account before buying stocks. Users can see their funds, and total_investing_value which is the total value of shares and funds.
- The platform contains many stocks. Each stock has a unique stock_symbol, and a unique stock_name. Each stock has a CEO_name, founded_year, and price, which describes the stock or company. A stock owned needs to be owned by someone.
- To sell or buy a stock, a user must post a request. A sale_requsetconsists of a unique sale_req_id, and price, stock_symbol, and quantity. A user can request, and request can only exist under one user. A user can also respond to a sale request by accepting them.

Functionality

- User

- Buy: When a user buys stock, they will see who is selling that stock for that company. The user can look at the selling price from other users in a list and choose whos to buy (most likely the cheapest). If the account balance is lower than the price of the stock, the purchase fails. Otherwise, subtract the stock price from the account balance for a successful purchase.
- Sell: When a user sells a stock, they create a sales request for that stock. The stock is taken out of their account and added to the request. The funds are then transferred to their account.

- Admin
  - Has all user functionality
  - Delete: Admins can delete sales and buy requests that they see as inappropriate or fishy. By sending a delete request, the database will perform a delete query and will delete the request.

Final Design of the Database

Tables and columns meanings and purpose

bank_account: stores user's bank information for adding funds

<u>user_id</u>	routing_number	account_number

user_id (numeric): foriegn key from stock_account.user)id, each and every user has an unique ID
routing_number (int): a user will save their routing number for adding funds
account_number(int): a user will save their account number for adding funds

stock_account: stores user account information

<u>user_id</u>	user_email	user_password	total_value	funds	user_name	is_admin

user_id (numeric): each and every user has an unique ID
user_email (varchar): user's email
user_password (hashed): user's password hashed in PHP
total_value (numeric): total value of account, calculated by funds + all share's current value
funds (numeric): total amount of money that user can use to buy shares
user_name (varchar): user's name that appears on webpage

is_admin (tinyint): defines if user is an admin: 0 = regular user, 1 = admin

sale_request: stores all sale requests

sale_req_id	user_id	price	stock_symbol	quantity	req_date	accept_date

sales_req_id (numeric): sales request comes along with an ID, each new post increments it’s ID by one

user_id (numeric): foriegn key from stock_account.user)id, each and every user has an unique ID

price (numeric): user assigns a price at which they want to sell their shares

stock_symbol (varchar): forien key from stock_info.stock_symboluser assigns what share they want to sell

quantity (numeric): user assigns how many shares they want to sell

req_date (date): stores the date and time the sale request is posted

accept_date (date): stores the date and time the sale request is accepted

stock_info: stores all stock informations

stock_symbol	stock_name	ceo	founded_year	current_value

stock_symbol (varchar): four letter symbol of the share

stock_name (varchar): the company name of the share

ceo (varchar): the CEO of the company

founded_year (varchar): the founded year of the company

current_value (varchar): current value of share of the company

shares: stores all shares owned by users

user_id	stock_symbol	purchase_date	current_value

user_id (numeric): foriegn key from stock_account.user)id, each and every user has an unique ID

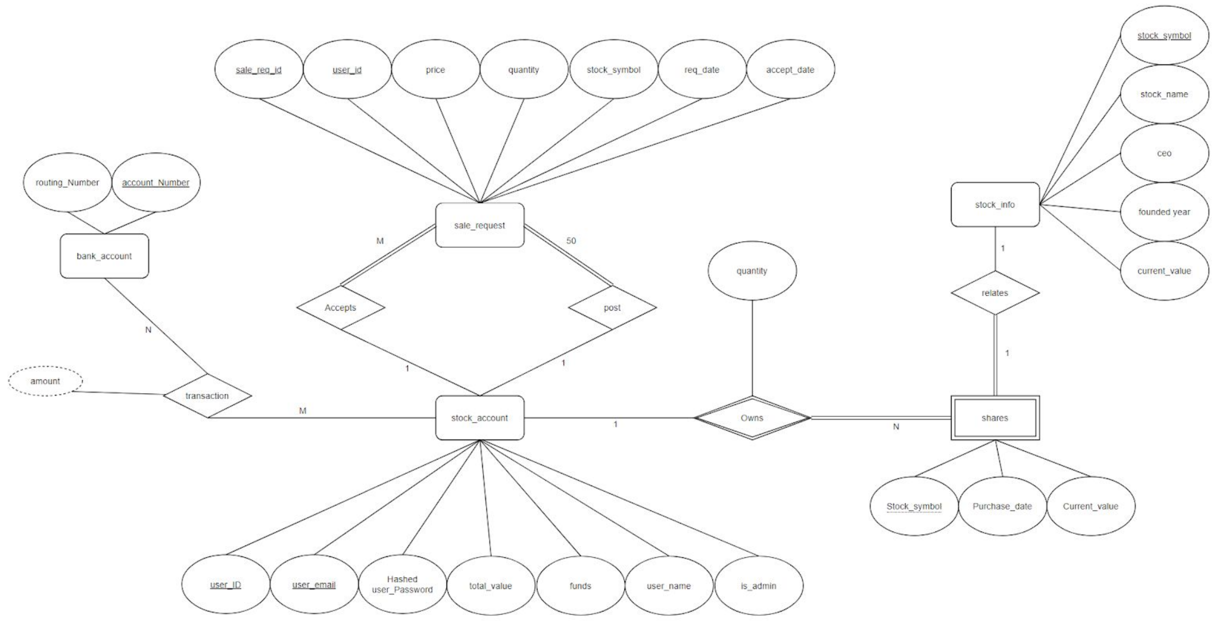
stock_symbol (varchar): forien key from stock_info.stock_symbol, four letter symbol of the share

purchase_date (date): forien key from sale_request.accept_date, date and time the sale request is accepted

current_value (varchar): forien key from stock_info.current_value, current value of the share

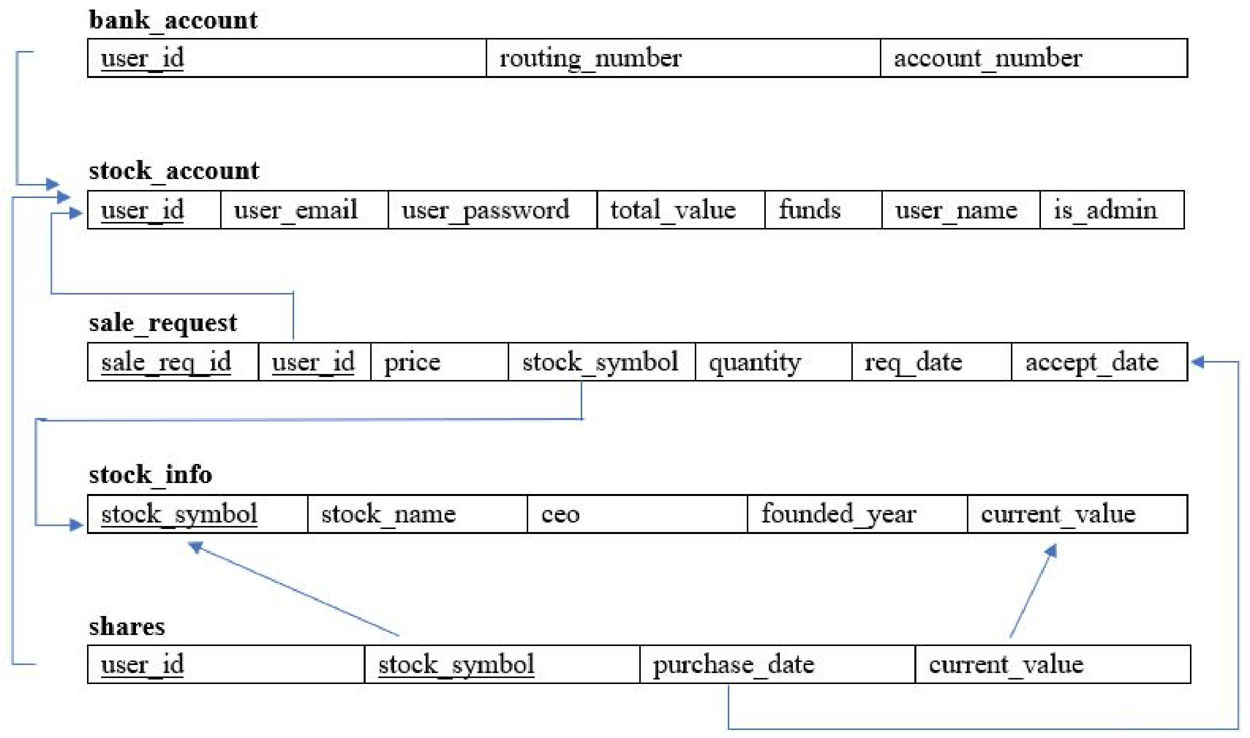
ER Diagram

The ER diagram we created is as shown below (Zoom in to see entity, relationship, and attribute).



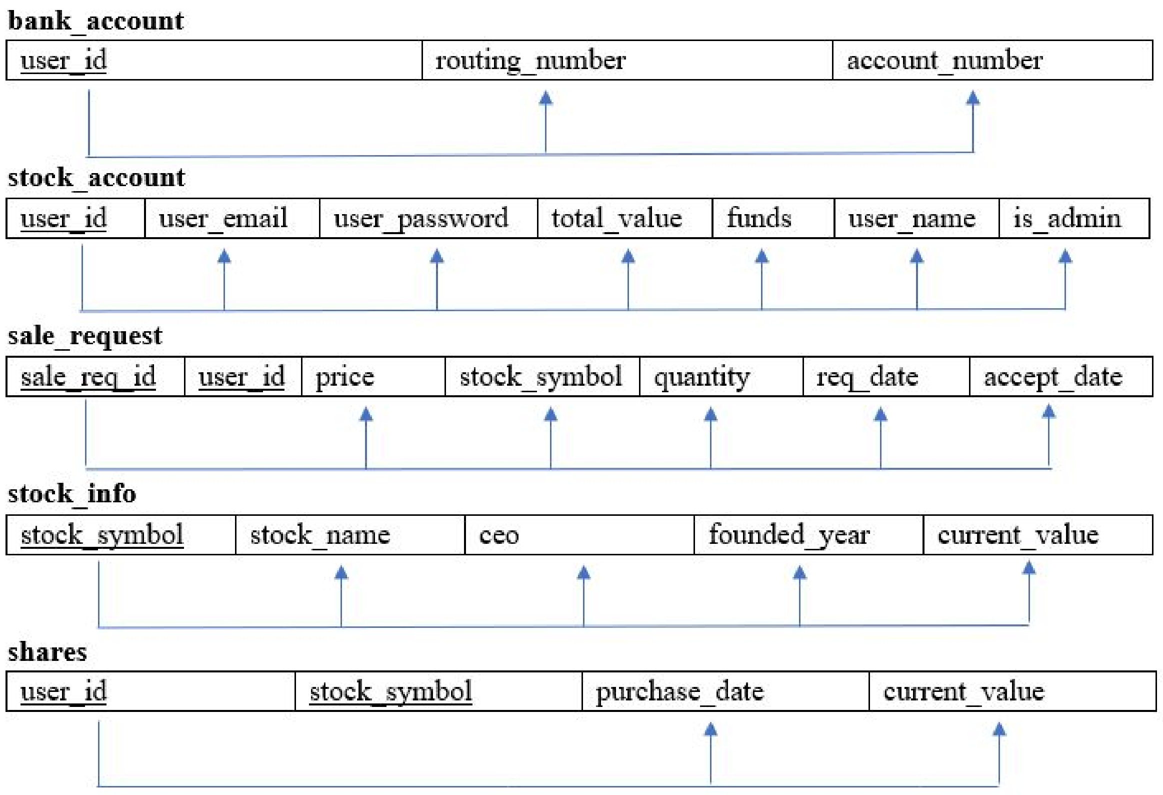
Data Schema

From our ER diagram, we created our data schema for our database. For our project, we were successfully able to create all our entities using 3NF normalization. Our data schema is shown in the diagram below.



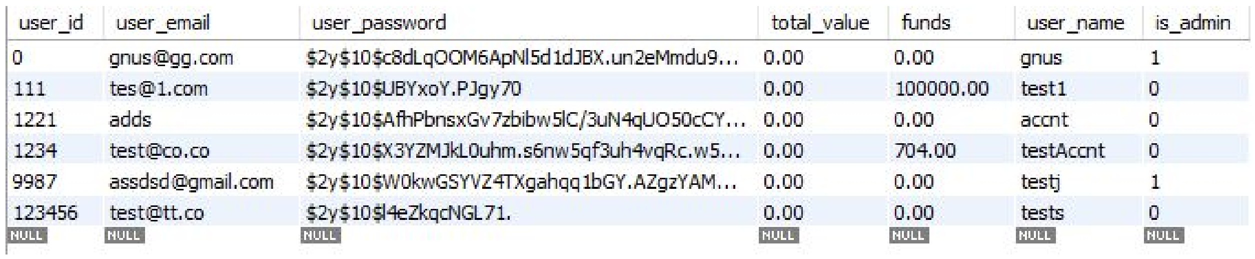
Functional dependencies of each table and normalization

Normalization: 3NF for all tables



User Login

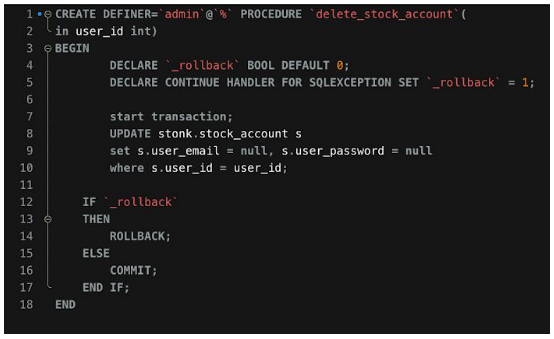
- The user login is hashed using the password_hash function in PHP. This creates a strong 1-way hash with salt for the users password.
- The password_verify function is used when the user logs in. This ensures that the password is not stored or sent to the database in plain text during the verification process when logging in.
- Sample user table with hashed password



user_id	user_email	user_password	total_value	funds	user_name	is_admin
0	gnus@gg.com	$2y$10$c8dLqOOM6ApNl5d1dJBX.un2eMmdu9...	0.00	0.00	gnus	1
111	tes@1.com	$2y$10$UBYxoY.PJgy70	0.00	100000.00	test1	0
1221	adds	$2y$10$AfhPbnsxGv7zbibw5lC/3uN4qUO50cCY...	0.00	0.00	accnt	0
1234	test@co.co	$2y$10$X3YZMJkL0uhm.s6nw5qf3uh4vqRc.w5...	0.00	704.00	testAccnt	0
9987	assdsd@gmail.com	$2y$10$W0kwGSYVZ4TXgahqq1bGY.AZgzYAM...	0.00	0.00	testj	1
123456	test@tt.co	$2y$10$l4eZkqcNGL71.	0.00	0.00	tests	0
NULL	NULL	NULL	NULL	NULL	NULL	NULL

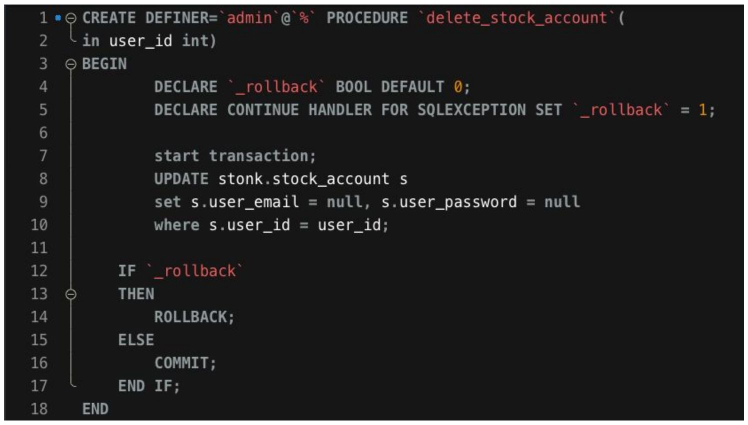
Stored procedures

- delete_bank_account(user_id)



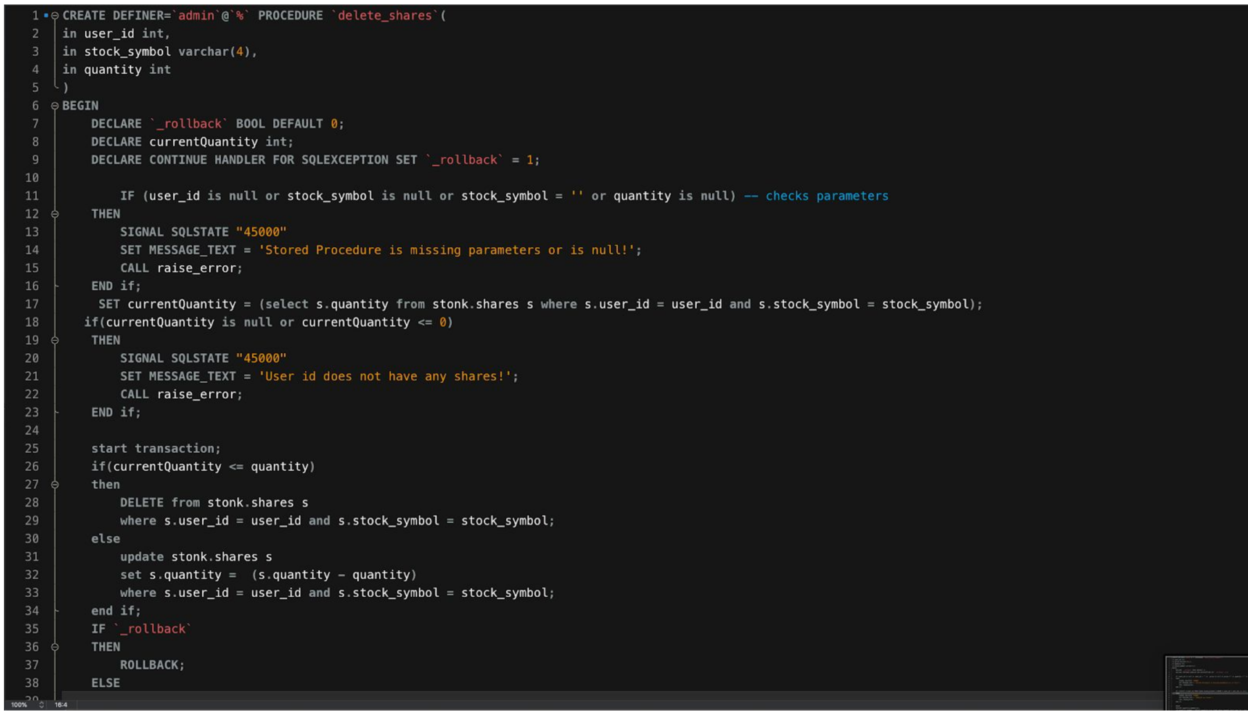
```
CREATE DEFINER=`admin`@`%` PROCEDURE `delete_stock_account`(
in user_id int)
BEGIN
        DECLARE `_rollback` BOOL DEFAULT 0;
        DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET `_rollback` = 1;

        start transaction;
        UPDATE stonk.stock_account s
        set s.user_email = null, s.user_password = null
        where s.user_id = user_id;

    IF `_rollback`
    THEN
        ROLLBACK;
    ELSE
        COMMIT;
    END IF;
END
```

- delete_stock_account(user_id)
    - For all other deletes, we actually delete the data. But for stock account, we decided to just set the password and username to null.



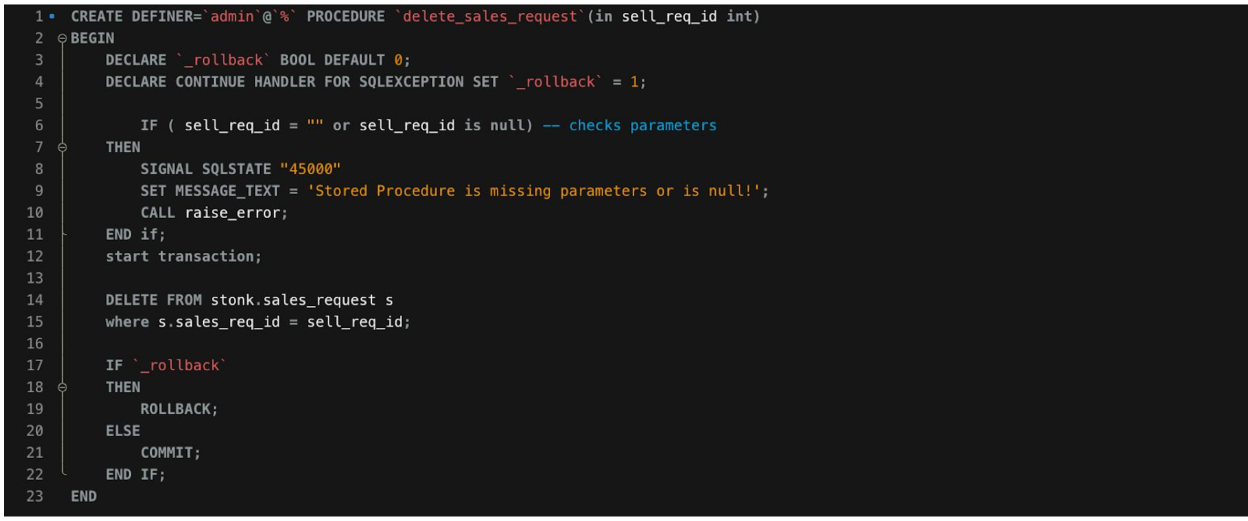
```
CREATE DEFINER=`admin`@`%` PROCEDURE `delete_stock_account`(
in user_id int)
BEGIN
        DECLARE `_rollback` BOOL DEFAULT 0;
        DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET `_rollback` = 1;

        start transaction;
        UPDATE stonk.stock_account s
        set s.user_email = null, s.user_password = null
        where s.user_id = user_id;

    IF `_rollback`
    THEN
        ROLLBACK;
    ELSE
        COMMIT;
    END IF;
END
```

- delete_shares(user_id, stock_symbol)



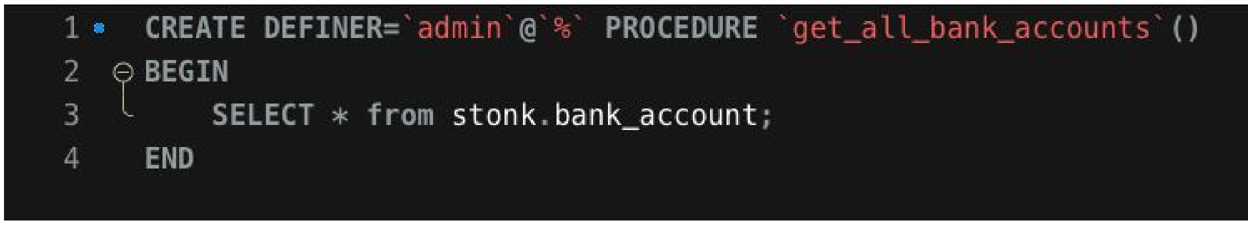
```
CREATE DEFINER=`admin`@`%` PROCEDURE `delete_shares`(
in user_id int,
in stock_symbol varchar(4),
in quantity int
)
BEGIN
    DECLARE `_rollback` BOOL DEFAULT 0;
    DECLARE currentQuantity int;
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET `_rollback` = 1;

        IF (user_id is null or stock_symbol is null or stock_symbol = '' or quantity is null) -- checks parameters
    THEN
        SIGNAL SQLSTATE "45000"
        SET MESSAGE_TEXT = 'Stored Procedure is missing parameters or is null!';
        CALL raise_error;
    END if;
     SET currentQuantity = (select s.quantity from stonk.shares s where s.user_id = user_id and s.stock_symbol = stock_symbol);
   if(currentQuantity is null or currentQuantity <= 0)
    THEN
        SIGNAL SQLSTATE "45000"
        SET MESSAGE_TEXT = 'User id does not have any shares!';
        CALL raise_error;
    END if;

    start transaction;
    if(currentQuantity <= quantity)
    then
        DELETE from stonk.shares s
        where s.user_id = user_id and s.stock_symbol = stock_symbol;
    else
        update stonk.shares s
        set s.quantity =  (s.quantity - quantity)
        where s.user_id = user_id and s.stock_symbol = stock_symbol;
    end if;
    IF `_rollback`
    THEN
        ROLLBACK;
    ELSE
```

- delete_sell_request(sell_request_id)



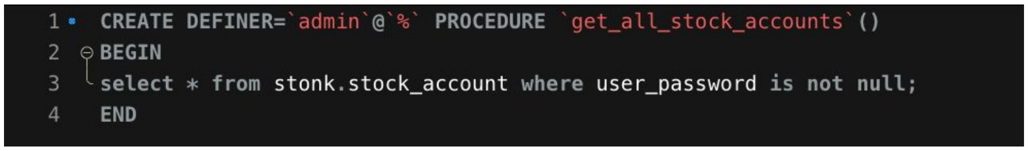
```
CREATE DEFINER=`admin`@`%` PROCEDURE `delete_sales_request`(in sell_req_id int)
BEGIN
    DECLARE `_rollback` BOOL DEFAULT 0;
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET `_rollback` = 1;

        IF ( sell_req_id = "" or sell_req_id is null) -- checks parameters
    THEN
        SIGNAL SQLSTATE "45000"
        SET MESSAGE_TEXT = 'Stored Procedure is missing parameters or is null!';
        CALL raise_error;
    END if;
    start transaction;

    DELETE FROM stonk.sales_request s
    where s.sales_req_id = sell_req_id;

    IF `_rollback`
    THEN
        ROLLBACK;
    ELSE
        COMMIT;
    END IF;
END
```

- get_all_bank_accounts()

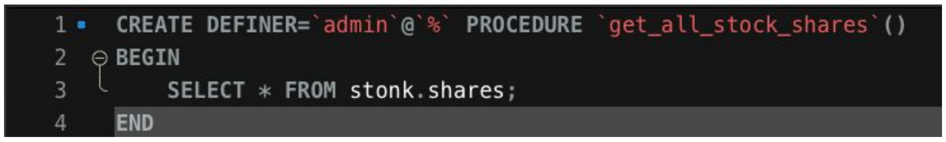


```
CREATE DEFINER=`admin`@`%` PROCEDURE `get_all_bank_accounts`()
BEGIN
    SELECT * from stonk.bank_account;
END
```

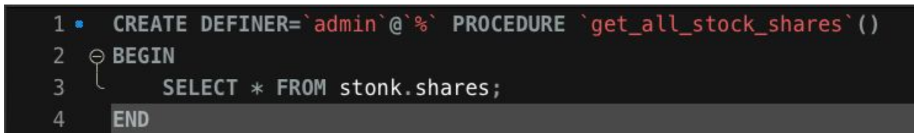
- get_stock_accounts()

- 

```
CREATE DEFINER=`admin`@`%` PROCEDURE `get_all_stock_accounts`()
BEGIN
select * from stonk.stock_account where user_password is not null;
END
```

- Get_all_stock_shares()
  - 

```
CREATE DEFINER=`admin`@`%` PROCEDURE `get_all_stock_shares`()
BEGIN
    SELECT * FROM stonk.shares;
END
```

- get_sell_requests()
  - 

```
CREATE DEFINER=`admin`@`%` PROCEDURE `get_all_stock_shares`()
BEGIN
    SELECT * FROM stonk.shares;
END
```

Final Design of Web DB Application

Functionality with multiple SQL queries

- Functionality involving accessing more than one table
  - When a user buys stocks from a sell request, they provide the request ID and the quantity. In order to update the shares table with their new purchase, the sales_request table must be accessed to retrieve the stock and quantity as well as price. This is used to verify the user has the funds as well as if the quantity needs to be updated, or if the accept date needs to be updated (which closes the sales request). This is necessary since the information is coming from the sales_request table, gets verified/manipulated using PHP, then gets stored into the shares table, as well as the stock_acount table since funds are only available in the stocks_account table.

Trade Offs

- A major trade off was functionality vs complexity. None of the group members have experience in web development, so it was a learning experience, and creating added complexity in the web application part of the project that wasn't necessary took away from the database portion.
- Another tradeoff was in the SQL queries themselves. We ran into some issues with calling nested queries with variables in the nested part from the php connector. Because

we needed the nested portion of the query to perform the final query as well as the result of the nested query to use in the PHP logic, we could not get the result of the inner query to store into a php variable so we designed it with each inner query storing the result into a php variable and then using those both in the php as well as passing it into the final sql query.

Major Modifications to Proposal and ERD

- The biggest modification we made was removing the buy_request functionality. It didn't make sense to have users create buy requests that a seller would then fulfill. Instead, we changed the design so that when a buyer wants to buy they find a sell request, and then fulfill it by purchasing the desired quantity from the sell request.
- We also modified the pricing so that it is a marketplace. We originally had the design as a market place but that prices would fall and rise as the stock value did. This contradicted another portion of the design where we specified that buyers could buy the lowest priced sell request. We ended up showing the market value of the stock on the Stock Info page instead and then allowing users to buy and sell at their own prices.
- The stock_account table was also modified to add a user name field. Logging in with a user ID was not very intuitive, and the user id simplified this process as well as makes the login process similar to most websites which use a username for login. User_name is a primary key for stock_account.

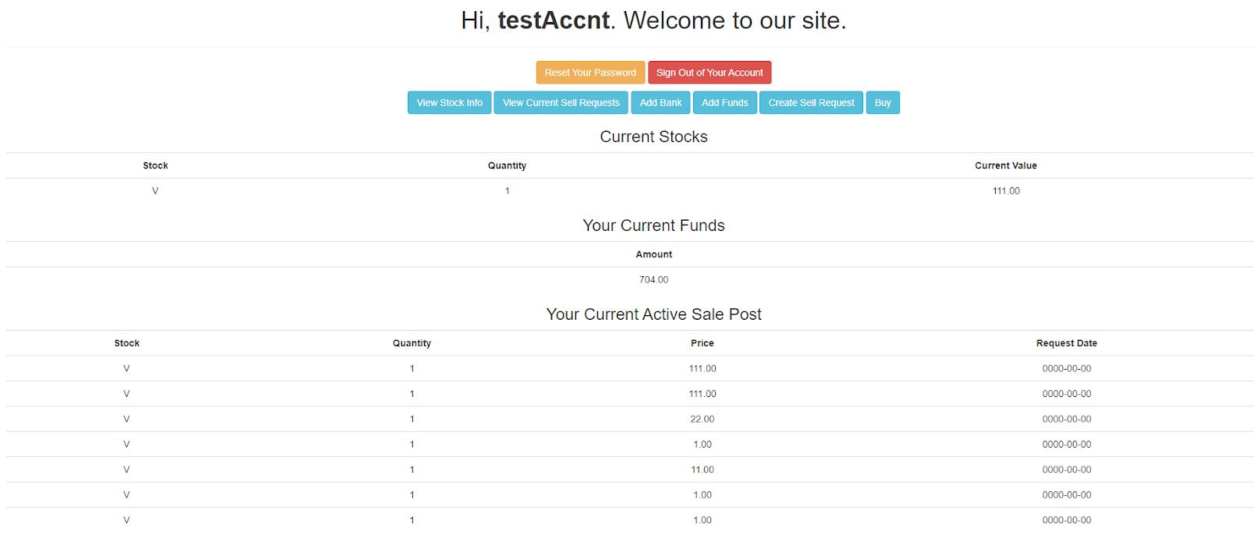
Functionality test cases and test plan execution

Test Plan: Each of the test cases will be run. Share data will be arbitrarily assigned to users, since it is impossible to create sell requests if no user owns any shares. All results are displayed with the test case. Any issues uncovered are in the postmortem section.

- Test Case 1: User registering: Test that a user can create an account, that if they don't fill in a required field it errors and that if they choose a duplicate user_id, it errors.
    - User account created, testAccnt. Login is verified.
    - Cannot create another account with the login.

- Test Case 2: Logging In: test that a user can log in, that they must fill in all fields, that if they don't have a valid username that is shown and that if the password is wrong, that is shown.
  - Login without username or password prompts an error
  - Login with a non existent user prompts an error: userid already exists
  - Login with a wrong password: prompts an error: invalid password
- Test Case 3: Sale requests: users are able to view sale requests that are active, as well as create sale requests for their shares. They must have the shares in their account to create the request
  - All active sales requests are shown, verified against the table in MySQL workbench
  - Creating a sales request is successful. The request shows up on the active sales request page as well as on their homepage. The shares are removed from their account.
- Test Case 4: Buy: User is able to buy shares. When they do, the quantity they bought is subtracted from the quantity listed in the sales request, unless they buy all of the shares in the request, in which case the request is closed.
  - Buying some of the shares from a sales request is successful. The shares are subtracted from the request. The shares are added to the user's shares.
  - Buying all of the shares from a sales request is successful. The shares are added to the user's shares and the request is closed (accept date filled with the current date)
- Test Case 5: User is able to view stock info: User is able to view all of the stock info for the stocks supported by the project.
  - View stock information such as company name and CEO on page is successful.
- Test Case 6: User is able to reset password
  - Password reset changes users password
  - Old password: unable to login, password invalid
  - New password: works as expected
- Test Case 7: User is able to log out
  - Logging out ends the session. Redirects user to login page.

Interface Screenshots



Hi, **testAccnt**. Welcome to our site.

Current Stocks

Stock	Quantity	Current Value
V	1	111.00

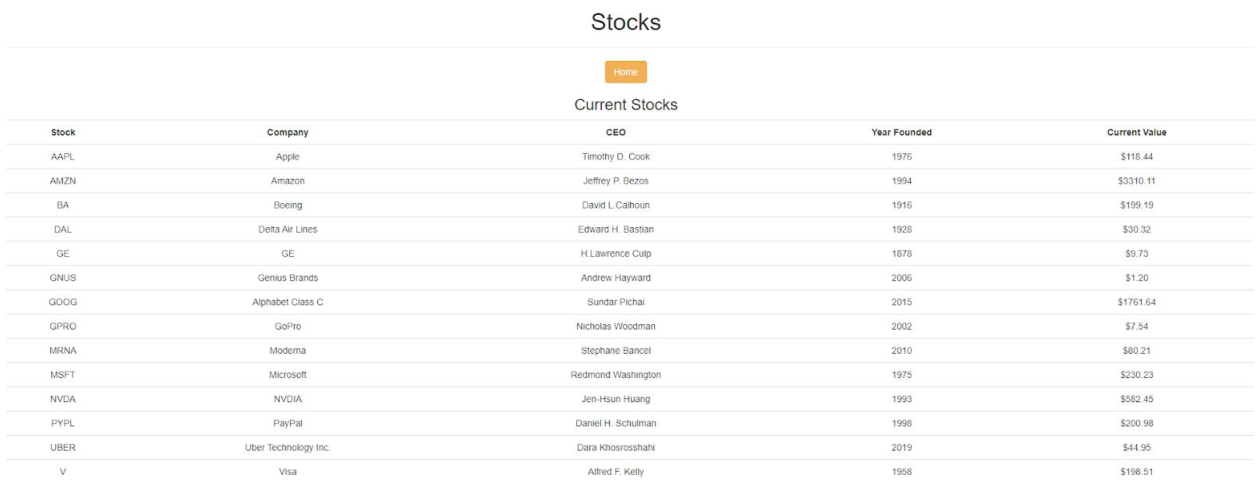
Your Current Funds

Amount
704.00

Your Current Active Sale Post

Stock	Quantity	Price	Request Date
V	1	111.00	0000-00-00
V	1	111.00	0000-00-00
V	1	22.00	0000-00-00
V	1	1.00	0000-00-00
V	1	11.00	0000-00-00
V	1	1.00	0000-00-00
V	1	1.00	0000-00-00

Figure 1: user welcome page

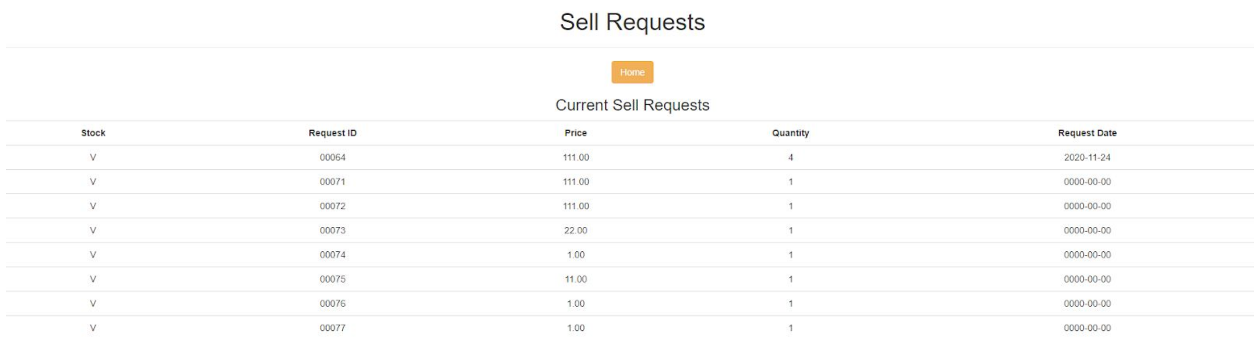


Stocks

Current Stocks

Stock	Company	CEO	Year Founded	Current Value
AAPL	Apple	Timothy D. Cook	1976	$118.44
AMZN	Amazon	Jeffrey P. Bezos	1994	$3310.11
BA	Boeing	David L.Calhoun	1916	$199.19
DAL	Delta Air Lines	Edward H. Bastian	1928	$30.32
GE	GE	H.Lawrence Culp	1878	$9.73
GNUS	Genius Brands	Andrew Hayward	2006	$1.20
GOOG	Alphabet Class C	Sundar Pichai	2015	$1761.64
GPRO	GoPro	Nicholas Woodman	2002	$7.54
MRNA	Moderna	Stephane Bancel	2010	$80.21
MSFT	Microsoft	Redmond Washington	1975	$230.23
NVDA	NVDIA	Jen-Hsun Huang	1993	$582.45
PYPL	PayPal	Daniel H. Schulman	1998	$200.98
UBER	Uber Technology Inc.	Dara Khosrosshahi	2019	$44.95
V	Visa	Alfred F. Kelly	1958	$198.51

Figure 2: Stock Info



Sell Requests

Current Sell Requests

Stock	Request ID	Price	Quantity	Request Date
V	00064	111.00	4	2020-11-24
V	00071	111.00	1	0000-00-00
V	00072	111.00	1	0000-00-00
V	00073	22.00	1	0000-00-00
V	00074	1.00	1	0000-00-00
V	00075	11.00	1	0000-00-00
V	00076	1.00	1	0000-00-00
V	00077	1.00	1	0000-00-00

Figure 3: Active Sell Request

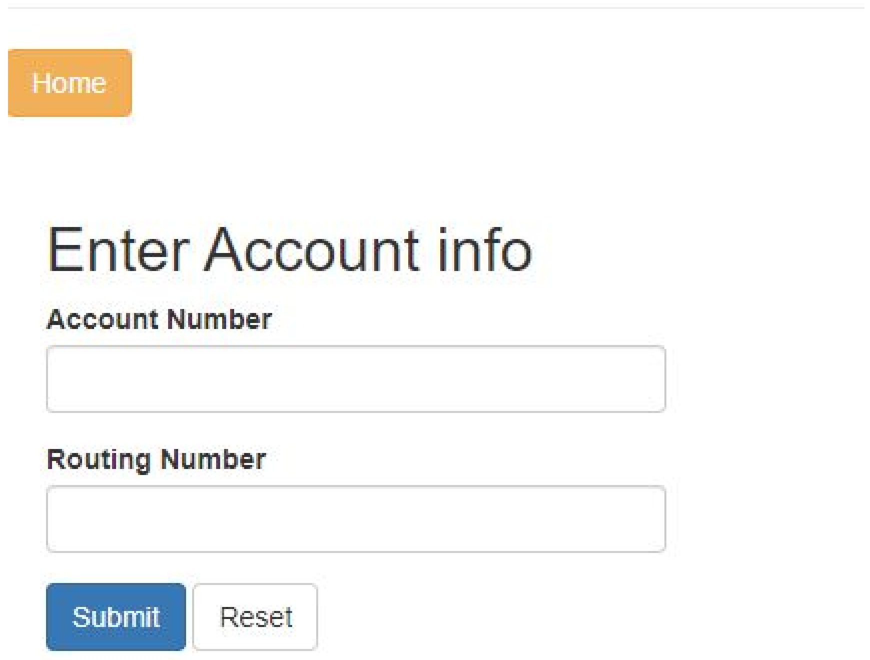


Figure 4: Enter Info for Bank Account

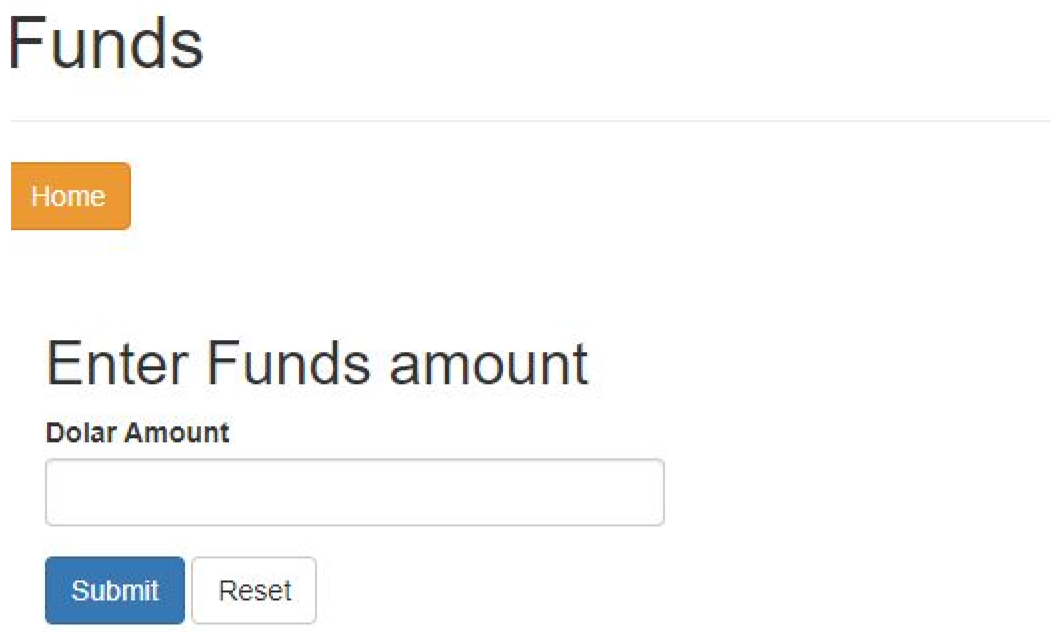
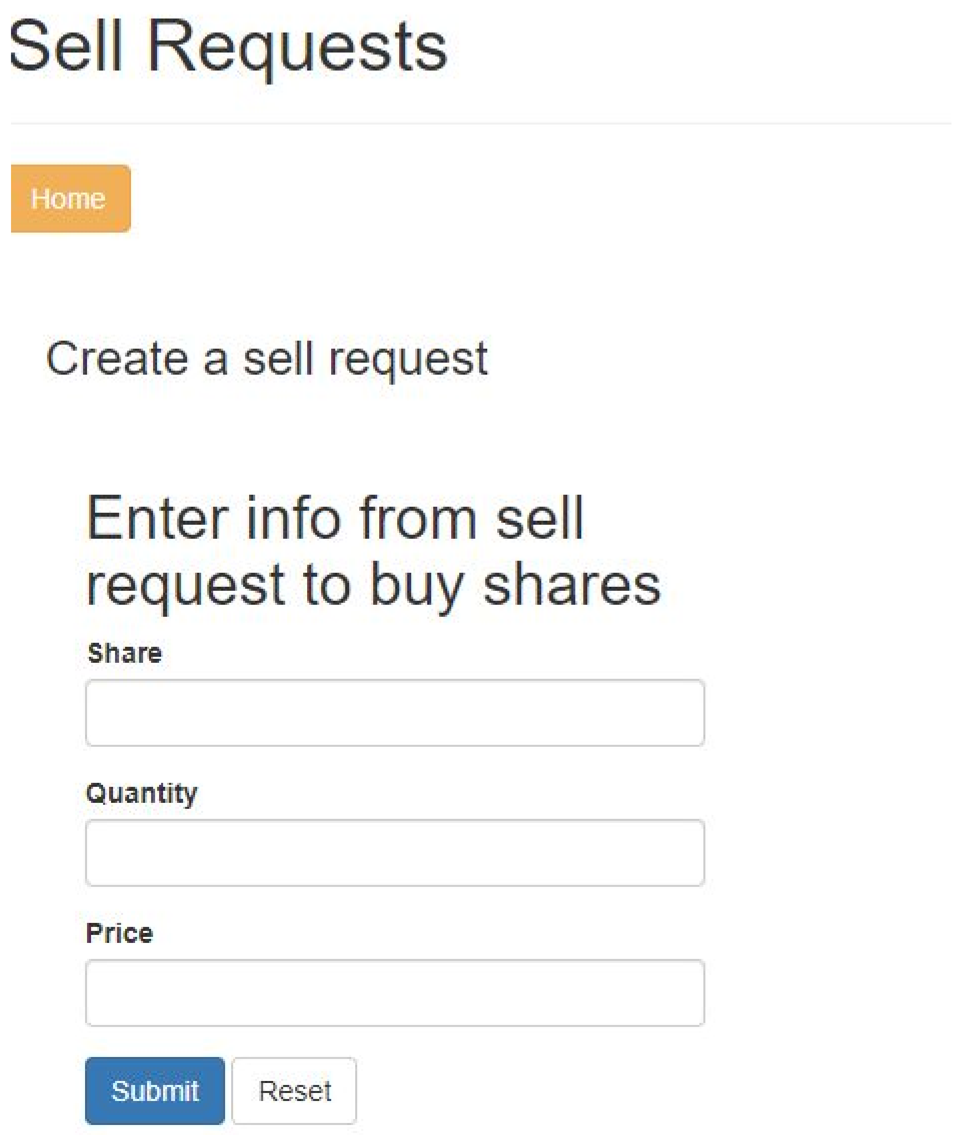


Figure 5: add funds



Sell Requests

Create a sell request

Enter info from sell request to buy shares

Share

Quantity

Price

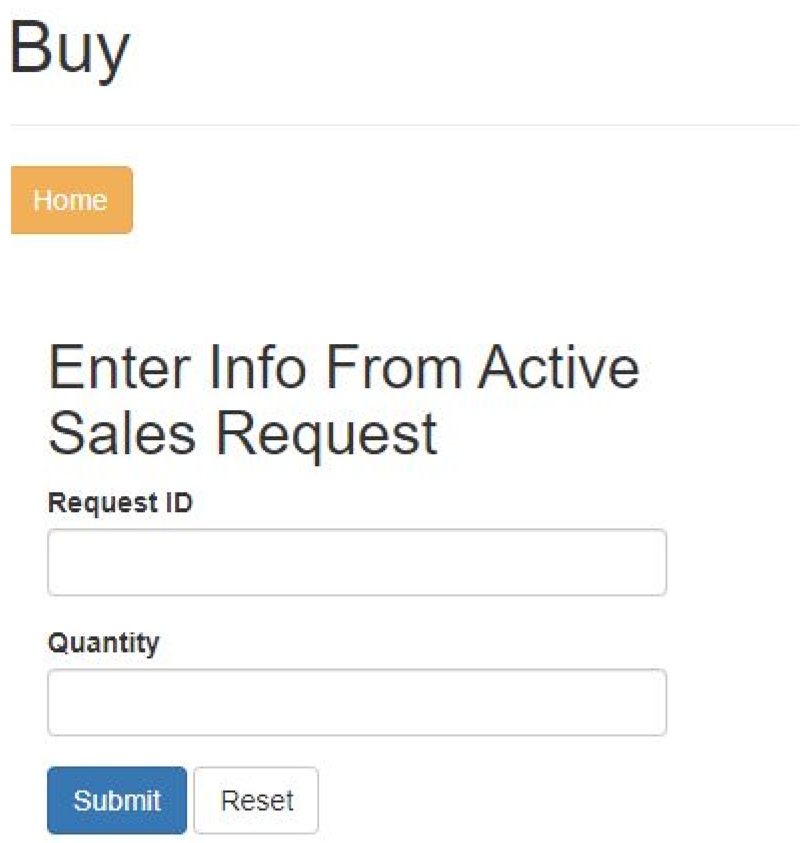
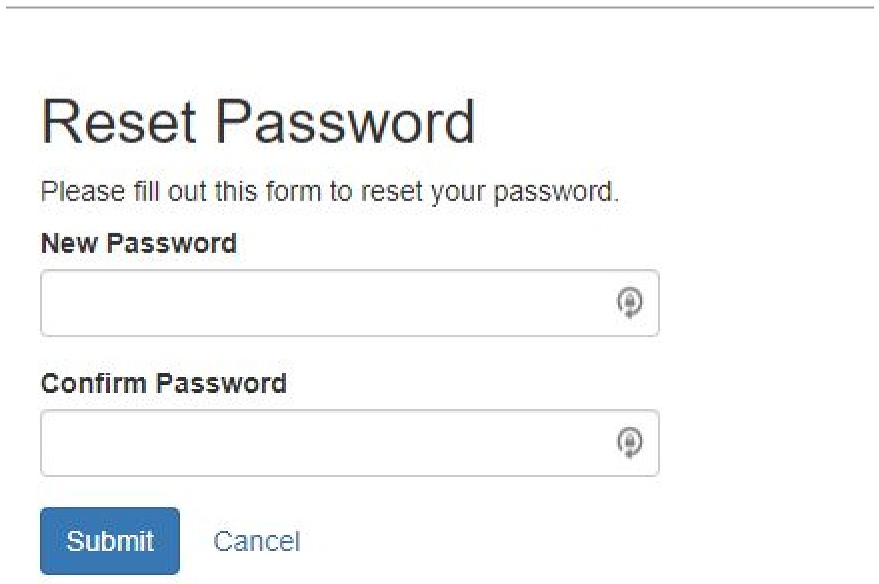

Buy
Home
Enter Info From Active Sales Request
Request ID
Quantity
Submit
Reset

Figure 6: Buy

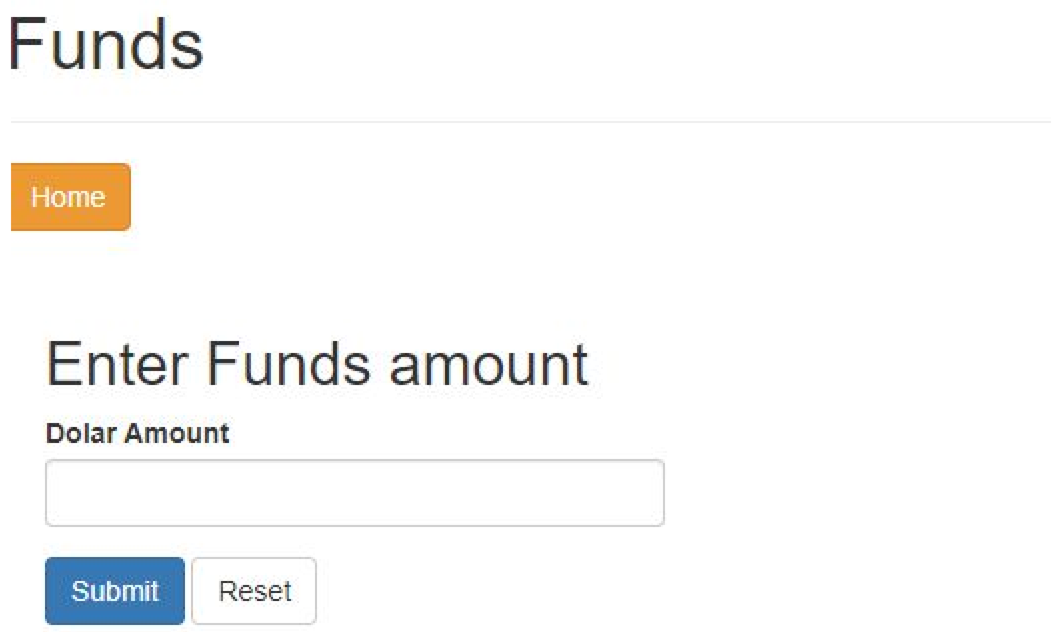

Reset Password

Please fill out this form to reset your password.

New Password

Confirm Password

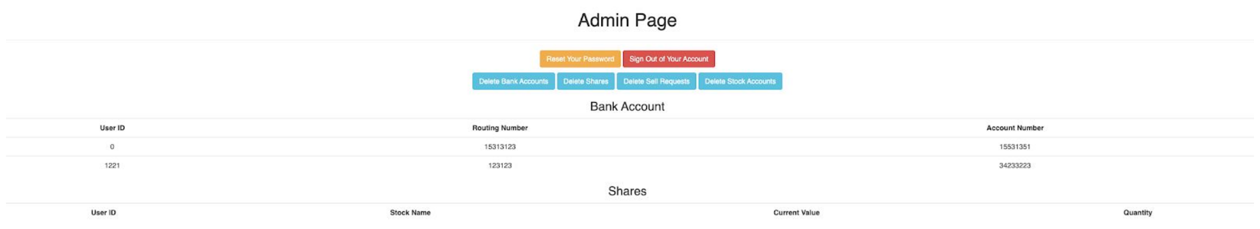
Figure 7: Reset password


Funds

Enter Funds amount

Dolar Amount

Figure 8: Enter Funds



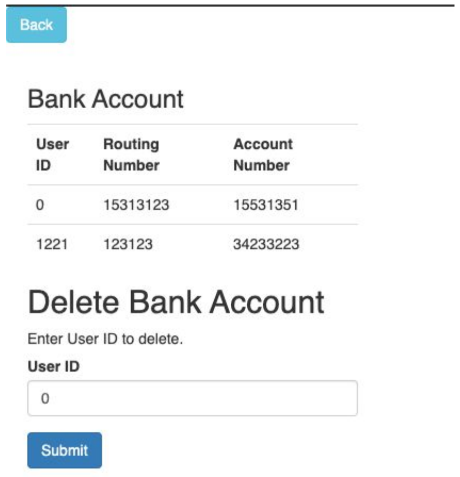
Bank Account

User ID	Routing Number	Account Number
0	15313123	15531351
1221	123123	34233223

Shares

User ID	Stock Name	Current Value	Quantity

Figure 9: Admin home page



Bank Account

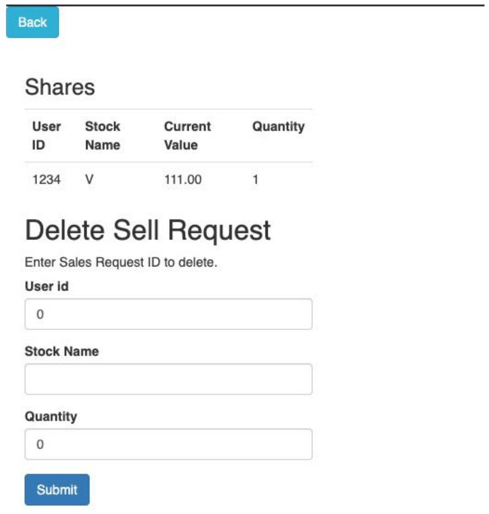
User ID	Routing Number	Account Number
0	15313123	15531351
1221	123123	34233223

Delete Bank Account

Enter User ID to delete.

User ID: 0

Figure 10: Delete Bank Account



Shares

User ID	Stock Name	Current Value	Quantity
1234	V	111.00	1

Delete Sell Request

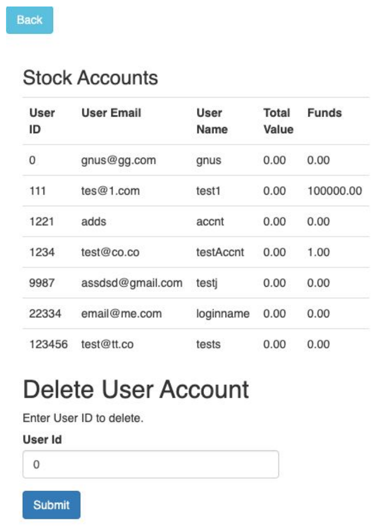
Enter Sales Request ID to delete.

User id: 0

Stock Name:

Quantity: 0

Figure 11: Admin delete sell request


Back

Stock Accounts

User ID	User Email	User Name	Total Value	Funds
0	gnus@gg.com	gnus	0.00	0.00
111	tes@1.com	test1	0.00	100000.00
1221	adds	accnt	0.00	0.00
1234	test@co.co	testAccnt	0.00	1.00
9987	assdsd@gmail.com	testj	0.00	0.00
22334	email@me.com	loginname	0.00	0.00
123456	test@tt.co	tests	0.00	0.00

Delete User Account

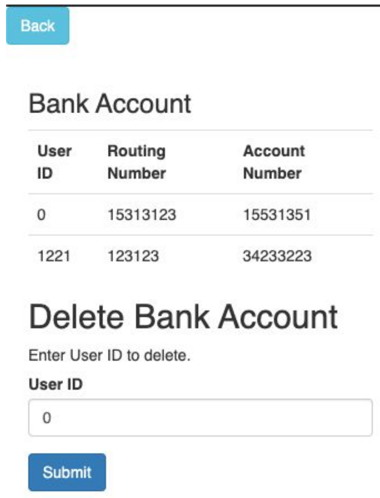
Enter User ID to delete.

User Id

0

Submit

Figure 12: Admin delete user page


Back

Bank Account

User ID	Routing Number	Account Number
0	15313123	15531351
1221	123123	34233223

Delete Bank Account

Enter User ID to delete.

User ID

0

Submit

Figure 13: Admin delete bank account page

Project Post Mortem

- **Issues**
    - Not all of the SQL queries are parameterized in the PHP, which could lead to a SQL injection attack. All of our put (insert, update, etc) statements are parameterized but a lot of the select statements are not.
    - User needs to sign out at the end session. If the user doesn't sign out before closing the application, the next time they sign in, they must sign out immediately and sign back in otherwise they won't see any of their data.

    - Two admin pages do not work currently
        - admin-delete-bank-account.php
        - admin-delete-stock-account .php
- **Improvements**
    - Parameterized select statements in PHP for added security.
    - Bank account should be tied to an API for the bank. Outside of the scope of the project though, since we are not implementing this with an actual bank account