CS 348 - Report

Kegan Allen, Luke Martin, Matteo Pesenti, Angelina Wu, Eric Zhang

Spring 2024 - Group 21

Link to repository: Poker Replay

R1 - Project Introduction

At present, our application will primarily be a central platform for poker players to view, comment on, and make use of their game replays and personal statistics, as well as those posted publicly by other users. As such, our database will be a repository of game results (play history, betting habits, etc.), loss/ gain tracking information, and account data.

To accomplish this, we must rely almost entirely on our users. Our focus on game statistics necessarily requires some level of input from clients, as obtaining the information directly from popular online poker platforms (PokerStars, etc.) requires technologies that are outside of the scope of this project.

R2 - Choice of Platform

Due to the relative variety and complexity of our data, it is needlessly complicated to interact with the database using the commandline. As such, our user interface will consist of a React-based website supported by a Flask back-end, giving us flexibility in how we manage the database while keeping the user experience simple. For the database, we intend on using PostgreSQL, so our local machines are our platform of choice.

The application is hosted on the cloud with DigitalOcean.

R3 - Sample Dataset

Link to sample dataset.

The sample data will be a subset of Matteo's actual poker hand histories. This data includes hand details, player information, dealt cards, betting actions, and the outcomes of each game. Users can request their hand history from PokerStars, which exports the data in a standardized format. We will use a parser to read the standardized files and extract the necessary information to prepare the data. The data will then be cleaned by normalizing formats using a parser and inserted into our database.

R4 - Production Dataset

Link to production dataset.

The production data will come from the full set of Matteo's hand histories. This dataset will grow over time as more users engage with the platform, as we will add support for uploading additional data. The dataset will include hand details, player information, specifics of each hand dealt, betting actions taken by players, and the outcomes of each game.

Users can request their hand histories from PokerStars and import them into the app.

R5 - Database Schema

R5a - Assumptions:

users:

- Each user has a unique username and email.

poker_session:

- Each poker_session has a corresponding user_id to reference an existing user.
- Poker_session has a unique tournament_id if that session is a tournament.
- Start_time is equal to the first poker hand of the session.
- End_time is the start of the last hand in the session.

poker_hand:

- Each record in poker_hand (session_id) is linked to a specific poker session (session_id).
- Each poker_hand has a unique site_hand_id.
- played_at field denotes the timestamp when the hand was played and is in UTC.

player:

- All players have a user_id linked to a tuple in the users table.

player_action:

- Each player action is associated with a specific hand and player.
- All specific actions (betting_round and action_type) and their valid values (Pre-flop, Flop, Turn, River for betting_round; fold, check, call, bet, raise, all-in for action_type) are correct in the provided data.
- Provided numerical values for each action are accurate.

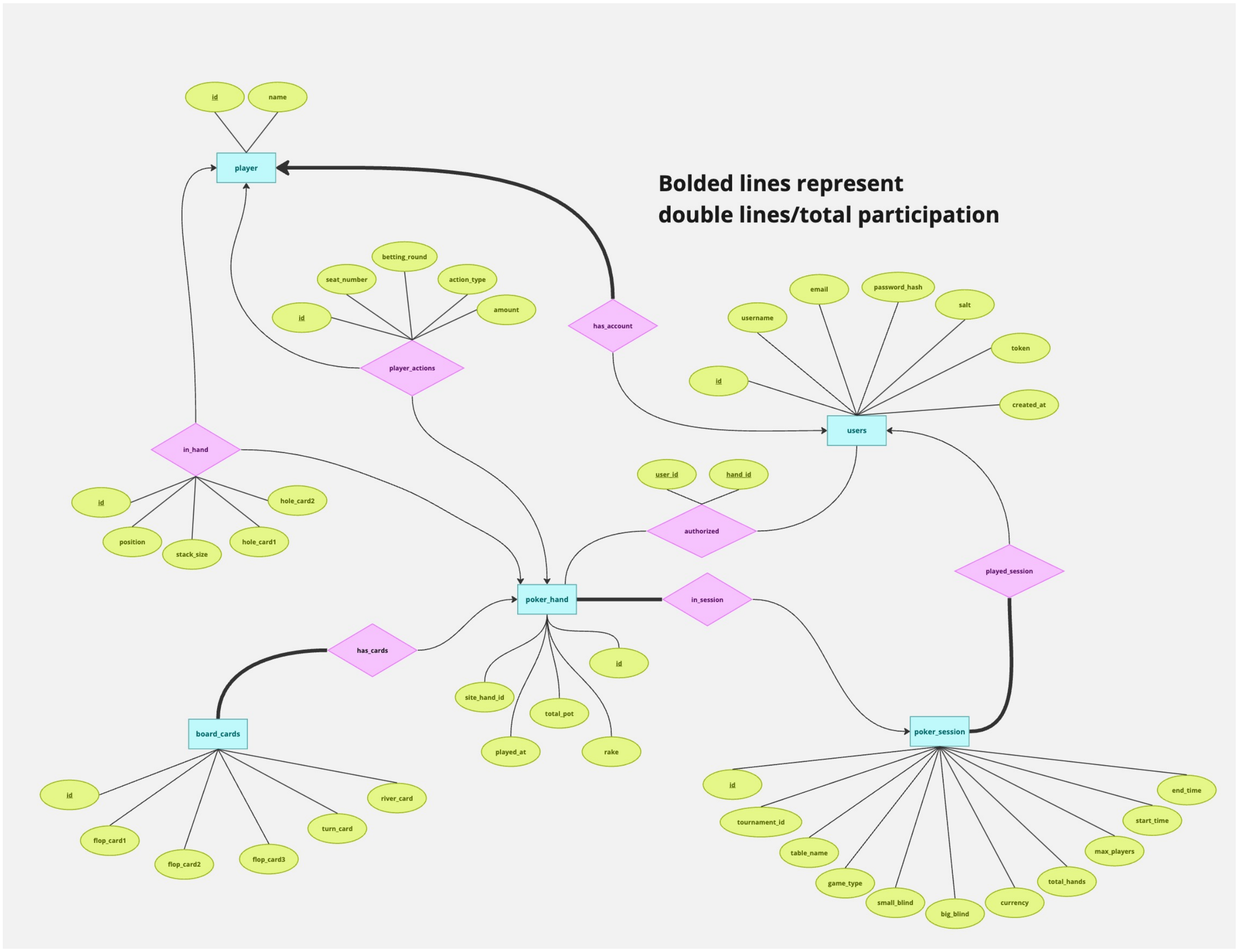
player_cards:

- Each record in player_cards is associated with a specific player and a specific hand.

board_cards:

- Each record in board_cards is associated with a specific poker hand.

R5b - E/R Diagram:



R5c - Database Table Declarations:

users(id, username, email, password_hash, salt, created_at, token)

poker_session(id, user_id, tournament_id, buy_in, table_name, game_type, currency, total_hands, max_players, start_time, end_time)

poker_hand(id, session_id, site_hand_id, small_blind, big_blind, total_pot, rake, played_at)

player(id, user_id, name)

player_action(id, player_id, hand_id, betting_round, action_type, amount)

player_cards(id, hand_id, player_id, hole_card1, hole_card2, position, stack_size)

board_cards(id, hand_id, flop_card1, flop_card2, flop_card3, turn_card, river_card)

authorized(user_id, hand_id)

R6-9 - Features

R6 - Hand Search

One of the core features of our application is the ability to search for and view information about any given hand that a user has played. While conceptually simple, the implementation will make use of every part of our technology stack. To find a played hand, its unique identifier must be given by the user in our web interface. From there, we can query our database for results from all of the relevant tables (player_action, player_cards, board_cards) to return a summary of information about a hand, the session it was played in, and the actions of each player. This data is then sent back to the front-end for viewing.

An example of how this feature will look can be seen in our database-driven application submitted with Milestone 1.

The queries used to implement this feature and expected outputs using our sample dataset are below.

```
PREPARE one_time_hand_info AS (
        SELECT poker_hand.id, poker_hand.session_id, poker_hand.total_pot,
               poker_hand.rake, poker_hand.played_at, poker_session.table_name,
               poker_session.game_type, poker_hand.small_blind, poker_hand.big_blind,
               board_cards.flop_card1, board_cards.flop_card2, board_cards.flop_card3,
               board_cards.turn_card, board_cards.river_card
        FROM poker_hand, poker_session, board_cards
        WHERE poker_hand.id = $1 AND poker_hand.session_id = poker_session.id
              AND board_cards.hand_id = $1
);

PREPARE player_actions_in_hand AS (
        SELECT player.id, player.name, player_action.id, player_action.hand_id,
               player_action.action_type, player_action.amount,
               player_action.betting_round
        FROM player, player_action
        WHERE player_action.hand_id = $1 AND player_action.player_id = player.id
);

PREPARE player_cards_in_hand AS (
        SELECT player.id, player.name, player_cards.hand_id, player_cards.hole_card1,
               player_cards.hole_card2, player_cards.position, player_cards.stack_size
        FROM player, player_cards
        WHERE player_cards.hand_id = $1 AND player_cards.player_id = player.id
);
```

with the selected data only coming from rows where the inputted hand_id matches with the queried tables. Each of the three queries fulfills a different role; the first searches for data that remains constant throughout a hand, the second returns a list of actions taken by each player, and the third queries for the cards each player is dealt. Additional contextual information is provided with each query as well to aid in preparing them for user consumption.

If we run each of the above queries on our sample dataset using a hand_id of 12, the results are:

```
one_time_hand_info:
id | session_id | total_pot | rake |      played_at      | table_name | game_type |  small_blind | big_blind | flop_card1 | flop_card2 | flop_card3 | turn_card | river_card
----+------------+-----------+------+---------------------+------------+-----------+-------------+-----------+------------+------------+------------+-----------+------------
 12 |          2 |      0.05 | 0.00 | 2023-09-27 23:05:04 | Alya III   | Cash      |  0.01 |      0.02 | 7h         | 2h         | Kc         |           |
```

```
player_actions_in_hand:
id |      name       | id  | hand_id | action_type | amount | betting_round
----+-----------------+-----+---------+-------------+--------+---------------
  1 | HortonRoundtree | 153 |      12 | ante        |   0.01 | Preflop
  2 | fourz4444       | 154 |      12 | ante        |   0.02 | Preflop
  3 | CashMatteo      | 155 |      12 | call        |   0.02 | Preflop
  6 | bouchizzle      | 156 |      12 | fold        |        | Preflop
  4 | ljab26          | 157 |      12 | fold        |        | Preflop
  5 | vegasricky      | 158 |      12 | fold        |        | Preflop
  1 | HortonRoundtree | 159 |      12 | fold        |        | Preflop
  2 | fourz4444       | 160 |      12 | check       |        | Preflop
  2 | fourz4444       | 161 |      12 | check       |        | Flop
  3 | CashMatteo      | 162 |      12 | bet         |   0.03 | Flop
  2 | fourz4444       | 163 |      12 | fold        |        | Flop
  3 | CashMatteo      | 164 |      12 | collect     |   0.05 | Showdown
```

```
player_cards_in_hand:
id |      name       | hand_id | hole_card1 | hole_card2 | position | stack_size
----+-----------------+---------+------------+------------+----------+------------
  1 | HortonRoundtree |      12 |            |            |        1 |       1.71
  2 | fourz4444       |      12 |            |            |        2 |       1.62
  3 | CashMatteo      |      12 |            |            |        3 |       2.41
  6 | bouchizzle      |      12 |            |            |        4 |       2.30
  4 | ljab26          |      12 |            |            |        5 |       2.60
  5 | vegasricky      |      12 |            |            |        6 |       1.71
```

R7 - Hand Visibility

All users of PokerReplay will have to log in. We want all users to control the visibility of the hands that they upload, so that they can be shareable via link, whitelisted, or private to themselves only. From the hand viewing page, the user can click a share button from which they can add a whitelist of usernames, click a checkbox to activate the link for sharing, or do nothing which maintains the default private visibility.

We have a table authorized that contains the id of the user and the id of the hand. When a user adds another user to a whitelist, we will insert the whitelisted user ID and the hand ID into the table. When the user sends an API request, we first check a their Bearer token against our records, then determine the owner (initial uploader) of the hand in question. If the request authorizes successfully and the owner is responsible for the request, only then will Authorized have the whitelisted user and hand ID inserted.

When requesting to get a certain hand ID, the id of the user making the request must be in the authorized table for that handId for any information to be returned.

The queries used to implement this feature and expected outputs using our sample dataset are below.

test_invalid.sql:

```
-- Incorrect case where user has no authorization
INSERT INTO authorized(user_id, hand_id)
SELECT 2, poker_hand.id FROM users
 JOIN poker_session ON users.id = poker_session.user_id
 JOIN poker_hand ON poker_session.id = poker_hand.session_id
 WHERE
   users.token = 'f273d736-807e-4f8e-b919-0bc7a558d59b'
   AND poker_hand.id = 9;
```

Expected Output:

```
INSERT 0 0
```

test_valid.sql:

```
-- Correct case where user does have authorization
INSERT INTO authorized(user_id, hand_id)
SELECT 2, poker_hand.id FROM users
 JOIN poker_session ON users.id = poker_session.user_id
 JOIN poker_hand ON poker_session.id = poker_hand.session_id
 WHERE
   users.token = '82fd7671-cfe6-49a6-b3b8-a0df504bf55f'
   AND poker_hand.id = 9;
```

Expected Output:

```
INSERT 0 1
```

test_view.sql:

```
-- Check all hands the user is authorized to see
SELECT poker_hand.id,
 poker_hand.session_id,
 poker_hand.site_hand_id,
 poker_hand.small_blind,
 poker_hand.big_blind,
 poker_hand.total_pot,
 poker_hand.rake,
 poker_hand.played_at
 FROM poker_session
 JOIN poker_hand ON poker_session.id = poker_hand.session_id
 JOIN authorized ON poker_hand.id = authorized.hand_id
 WHERE poker_session.user_id = 2 OR poker_hand.id IN
   (SELECT hand_id FROM authorized WHERE user_id = 2);
```

Expected Output:

```
 id | session_id | site_hand_id | small_blind | big_blind | total_pot | rake |   played_at
----+------------+--------------+-------------+-----------+-----------+------+---------------------
  9 |          1 | 246222408874 |        0.01 |      0.02 |      0.11 | 0.00 | 2023-09-27 23:03:22
(1 row)
```

R8 - Profit/Loss by User

This feature is for registered users who have populated one or more sessions of hand history. The intention is to have a display window to show various players by name, and include how much money has been gained or lost to that person. It will include multiple methods of sorting, so by clicking on the column header they can see the players they have gained the most off of, and lost the most to.

For any particular player, it sums the amount they bet on rounds you collected, and subtracts the total amount of money that you bet on rounds they collected. This way you can see who your money is going to.

The queries used to implement this feature and expected outputs using our sample dataset are below.

profit_loss_by_user.sql

```
 -- find the hands where the player_id plays
WITH user_player AS (
SELECT id FROM player WHERE player.user_id = '1'
),
-- all hands from cash sessions you own
hands AS (
SELECT id
FROM poker_hand hand
WHERE EXISTS (
     SELECT 1 FROM poker_session
     WHERE poker_session.id = hand.session_id
         AND poker_session.user_id = '1'
         AND poker_session.tournament_id IS NULL
)
),
-- map each hand to the player that collected it
collected_hands AS (
SELECT hand.id AS hand_id, (
        SELECT act.player_id
        FROM player_action act
        WHERE act.hand_id = hand.id
```

```
            AND act.action_type = 'collect'
        LIMIT 1
    ) AS collector_id
FROM hands hand
),
-- the total amount that each person contributed to each hand
bet_amounts AS (
SELECT hand_id, player_id, SUM(amount) as amount
FROM player_action
WHERE action_type in ('call', 'bet', 'raise', 'all-in')
GROUP BY hand_id, player_id
),
-- cash flow by hand and player to/from
cash_flow as (
SELECT
     -- relevant player, the non-user player involved
     (CASE
         WHEN hand.collector_id = user_player.id
         THEN bet_amounts.player_id

         WHEN bet_amounts.player_id = user_player.id
         THEN hand.collector_id

         ELSE NULL
     END) as player_id,
     -- raw cash flow
     CASE
-- when we win, add the amount we won from this player
         WHEN hand.collector_id = user_player.id
         THEN bet_amounts.amount

-- when someone else wins, if we bet, subtract the amount we lost to this player
         WHEN bet_amounts.player_id = user_player.id
         THEN -bet_amounts.amount

-- if we didn't win or place this bet, disregard the value
         ELSE 0
     END as amount
FROM user_player, collected_hands hand
NATURAL JOIN bet_amounts
)
SELECT player_id, SUM(amount)
FROM user_player, cash_flow
WHERE player_id <> user_player.id
GROUP BY player_id
```

Expected Output:

```
 player_id |  sum
-----------+-------
         1 |  0.10
         2 |  0.21
         4 |  0.48
         5 |  0.00
         6 | -0.06
```

R9 - Cash Flow by Hand

This query returns a full table of the user's hands over time, tracking how much they gained or lost during each of those hands. This is going to be used for a primary display used on the Figma to take the history of hands and when you gained or lost money, so you can quickly find notable games. For example, you may want to quickly find the ID of a hand you won big so you can share it with a friend, and this query supports that display on the front end.

The queries used to implement this feature and expected outputs using our sample dataset are below.

cash_flow_by_hand.sql:

```
-- calculate change in money over each hand
WITH user_player AS (
SELECT id FROM player WHERE player.user_id = 1 -- inject
),
hands AS (
    SELECT hand.id id, played_at
    FROM poker_session session
JOIN poker_hand hand ON session.id = hand.session_id
    WHERE user_id = 1 AND session.game_type = 'Cash' -- inject
),
bet_amounts AS (
    SELECT hand_id, SUM(
     CASE
     WHEN action_type = 'collect' THEN amount
     ELSE -amount
     END
    ) as amount
    FROM user_player, player_action
    JOIN hands ON hands.id = player_action.hand_id
    WHERE player_id = user_player.id
    GROUP BY hand_id, player_id
)
SELECT played_at, hand.id, COALESCE(SUM(amount), 0) amount
FROM hands hand
JOIN bet_amounts on hand.id = bet_amounts.hand_id
GROUP BY hand.id, played_at
ORDER BY played_at
```

Expected Output:

```
   played_at    |  id  | amount
---------------------+------+--------
 2023-09-27 22:56:41 | 1 |  -0.02
 2023-09-27 22:57:21 | 2 |  -0.01
 2023-09-27 22:58:09 | 3 |  -0.04
 2023-09-27 22:58:38 | 4 |   0
 2023-09-27 22:59:28 | 5 |  -0.02
 2023-09-27 23:00:26 | 6 |  -0.01
 2023-09-27 23:00:49 | 7 |   0.40
 2023-09-27 23:02:22 | 8 |  -0.02
 2023-09-27 23:03:22 | 9 |   0
 2023-09-27 23:04:06 |   10 |   0
 2023-09-27 23:04:33 |   11 |   0.00
 2023-09-27 23:05:04 |   12 |   0.00
 2023-09-27 23:05:51 |   13 |  -0.08
 2023-09-27 23:06:35 |   14 |  -0.01
 2023-09-27 23:06:57 |   15 |  -0.06
 2023-09-27 23:07:30 |   16 |   0.33
 2023-09-27 23:08:03 |   17 |   0
 2023-09-27 23:08:27 |   18 |   0.00
 2023-09-27 23:08:39 |   19 |  -0.04
 2023-09-27 23:09:44 |   20 |  -0.01
```

R12-14 - Fancy Features

R12 - Hand Parser

We developed a parser for the raw text provided to the online poker site PokerStars upon a data request.

```
PokerStars Hand #246222355402:
Hold'em No Limit ($0.01/$0.02 CAD) - 2023/09/27 22:56:41 ET
Table 'Alya III' 6-max Seat #1 is the button
Seat 1: HortonRoundtree ($1.86 in chips)
Seat 2: fourz4444 ($1.91 in chips)
Seat 3: CashMatteo ($2 in chips)
Seat 5: ljab26 ($2.96 in chips)
Seat 6: vegasricky ($1.56 in chips)
fourz4444: posts small blind $0.01
CashMatteo: posts big blind $0.02
bouchizzle: sits out
*** HOLE CARDS ***
Dealt to CashMatteo [8c 3d]
ljab26: raises $0.02 to $0.04
```

```
vegasricky: folds
HortonRoundtree: folds
fourz4444: calls $0.03
CashMatteo: folds
*** FLOP *** [4h Jd Kh]
fourz4444: checks
ljab26: bets $0.04
fourz4444: raises $0.18 to $0.22
ljab26: folds
Uncalled bet ($0.18) returned to fourz4444
fourz4444 collected $0.17 from pot
fourz4444: doesn't show hand
*** SUMMARY ***
Total pot $0.18 | Rake $0.01
Board [4h Jd Kh]
Seat 1: HortonRoundtree (button) folded before Flop (didn't bet)
Seat 2: fourz4444 (small blind) collected ($0.17)
Seat 3: CashMatteo (big blind) folded before Flop
Seat 5: ljab26 folded on the Flop
Seat 6: vegasricky folded before Flop (didn't bet)
```

The parser extracts critical information from a poker hand history that can be attained by requesting information from a poker app such as poker stars. Key functionalities of the parser include:

- **Position Tracking**: The parser determines the seating position of each player and inserts this information into the database.

- **Action Logging**: All player actions (e.g., bets, folds, raises) are parsed and stored in a player_actions table, providing a detailed record of each hand.

- **Card Recording**: The parser captures and stores revealed player cards and board cards, enabling comprehensive hand analysis.

This Python script ensures all participants in an online poker hand are identified and added to the database as necessary. This functionality supports our feature in R8 that tracks financial interactions with other players, identifying gains and losses.

The data provided by PokerStars is in a human-readable format, making extraction challenging. We accurately extracted the necessary information to fit our relational data model by employing regular expressions (RegEx).

The parser is versatile, working with both tournament and cash games, allowing us to track performance across different session types.

R13 - Password Security

Our users table stores hashed and salted passwords, as well as a token. A hash is a string that is generated from the plaintext password. The generation of the hash is done through an algorithm that is incredibly computationally expensive to reverse. If our database were to be compromised, the hash would obscure what the user's password is. This can prevent users' passwords from being exposed, especially in the case of when users use the same or similar passwords across multiple sites.

However, an attacker might have a list of common passwords converted to hashes. This creates a dictionary of common password hashes, without having to reverse the hashing algorithm itself. This is known as a rainbow table attack. However, we do a process known as salting the password of the user. It is a separate parameter added to compute the hash in addition to the password and is unique to each user. This second parameter changes the result of the hash function. Now, a rainbow table attack would need to recompute common passwords for each salt in the database to find a match. This increases the difficulty of cracking an individual user's password by a factor of n, where n is the number of users. Our hashing algorithm is bcrypt, which is an industry-standard for preventing rainbow table attacks.

This feature is present in our database schema through the password_hash and salt attributes:

```
CREATE TABLE users (
   id SERIAL PRIMARY KEY,
   username VARCHAR(50) UNIQUE NOT NULL,
   email VARCHAR(100) UNIQUE,
   password_hash CHAR(60), -- Assuming bcrypt hash which outputs 60 characters
   salt CHAR(29), -- Storing bcrypt salt, which is 29 characters long
   created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
   token CHAR(36)
);
```

R14 - UI/UX Design

Mockups of the web app were created using Figma (see here) to communicate the design and how the user will interact with the app. The current design uses minimalistic UI components, icons, and a cohesive color scheme. This makes it straightforward to translate from design to code, and is intuitive for the user to navigate. Relevant data is also displayed in tables for better organization and readability. The UI will continue to be updated as more features get added.

R17 - Group Member Contribution

Milestone 0

All group members participated in the preparation of Milestone 0 equally via writing or editing the report or the "hello world" equivalent.

Milestone 1

Kegan Allen - Completed the relevant sections for R8 and R9, worked on setting up cloud hosting, was a major collaborator on the E/R diagram, and Linux :).

Luke Martin - Completed the relevant sections for R6, worked on the back-end server, helped integrate back- and front-ends for simple application, and compiled the report.

Matteo Pesenti - Implemented the data parser, provided the current dataset, helped complete the relevant sections for R9, and designed current database schema.

Angelina Wu - Designed user interface, implemented current front-end interface, was a major collaborator on the E/R diagram.

Eric Zhang - Completed the relevant sections for R7, implemented password security, helped integrate back- and front-ends for simple application.

All members contributed to writing and editing various other sections of the Milestone 1 report.