SQL Creation

```
        // Ensure the users table exists
        pqxx::work W(*g_conn);
        W.exec(
            "CREATE TABLE IF NOT EXISTS users ("
            "   id SERIAL PRIMARY KEY, "
            "   username VARCHAR(50) UNIQUE NOT NULL, "
            "   password VARCHAR(255) NOT NULL"
            ");"
        );

        W.exec(
            "CREATE TABLE IF NOT EXISTS portfolios ("
            "   id SERIAL PRIMARY KEY, "
            "   user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE
CASCADE, "
            "   name VARCHAR(64) NOT NULL,  "
            "   cash INTEGER NOT NULL DEFAULT 0"
            ");"
        );

        W.exec(
            "CREATE TABLE IF NOT EXISTS holdings ("
            "   id SERIAL PRIMARY KEY, "
            "   portfolio_id INTEGER NOT NULL REFERENCES portfolios(id) ON
DELETE CASCADE, "
            "   symbol VARCHAR(16) NOT NULL, "
            "   quantity INTEGER NOT NULL DEFAULT 0, "
            "   UNIQUE (portfolio_id, symbol)"
            ");"
        );

        // friendship table is mutual
        W.exec(
            "CREATE TABLE IF NOT EXISTS friendships ("
            "   id SERIAL PRIMARY KEY, "
            "   user1_id INTEGER NOT NULL REFERENCES users(id) ON DELETE
CASCADE, "
            "   user2_id INTEGER NOT NULL REFERENCES users(id) ON DELETE
CASCADE, "
```

```
            "   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, "
            "   UNIQUE (user1_id, user2_id), "
            "   CHECK (user1_id < user2_id)"  // for consistent ordering
            ");"
        );

        W.exec(
            "CREATE TABLE IF NOT EXISTS friend_requests ("
            "   id SERIAL PRIMARY KEY, "
            "   from_user_id INTEGER NOT NULL REFERENCES users(id) ON
DELETE CASCADE, "
            "   to_user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE
CASCADE, "
            "   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, "
            "   rejected_at TIMESTAMP DEFAULT NULL, "
            "   UNIQUE (from_user_id, to_user_id), "
            "   CHECK (from_user_id != to_user_id)"
            ");"
        );


        //for stocklists visibility:0 public,1 private,2 shared
        W.exec(
            "CREATE TABLE IF NOT EXISTS stocklists ("
            "   id SERIAL PRIMARY KEY, "
            "   user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE
CASCADE, "
            "   name VARCHAR(64) NOT NULL, "
            "   visibility INTEGER NOT NULL DEFAULT 0"
            ");"
        );

        W.exec(
            "CREATE TABLE IF NOT EXISTS stocklist_to_user ("
            "   stocklist_id INTEGER NOT NULL REFERENCES stocklists(id) ON
DELETE CASCADE, "
            "   user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE
CASCADE, "
            "   UNIQUE (stocklist_id, user_id)"
            ");"
```

```
        );

        W.exec(
            "CREATE TABLE IF NOT EXISTS stocklist_holdings ("
            "   id SERIAL PRIMARY KEY, "
            "   stocklist_id INTEGER NOT NULL REFERENCES stocklists(id) ON
DELETE CASCADE, "
            "   quantity INTEGER NOT NULL DEFAULT 0, "
            "   symbol VARCHAR(16) NOT NULL, "
            "   UNIQUE (stocklist_id, symbol)"
            ");"
        );
        W.exec(
            "CREATE TABLE IF NOT EXISTS stocklist_descriptions ("
            "   stocklist_id INTEGER NOT NULL REFERENCES stocklists(id) ON
DELETE CASCADE, "
            "   description VARCHAR(4096) DEFAULT ''"
            ");"
        );
        W.exec(
            "CREATE TABLE IF NOT EXISTS stocklist_reviews ("
            "   stocklist_id INTEGER NOT NULL REFERENCES stocklists(id) ON
DELETE CASCADE, "
            "   reviewer_id INTEGER NOT NULL REFERENCES users(id) ON
DELETE CASCADE, "
            "   review VARCHAR(4096) DEFAULT '', "
            "   UNIQUE(stocklist_id, reviewer_id)"
            ");"
        );
```

Checking Credentials

```
pqxx::result R = W.exec_params(
            "SELECT id FROM users WHERE username = $1 AND password = $2",
            username,
            password
        );
```

Register Users

```
 W.exec_params(
            "INSERT INTO users (username, password) VALUES ($1, $2)",
            username,
```

```
            password
        );
```

LOGIN RELATED

Get User ID

```
        pqxx::result R = W.exec_params(
            "SELECT id FROM users WHERE username = $1",
            username
            );
```

Get Username

```
pqxx::result R = W.exec_params(
            "SELECT username FROM users WHERE id = $1",
            user_id
        );
```

PORTFOLIO RELATED

Get Portfolio

```
        pqxx::result R = W.exec_params(
            "SELECT id, name, cash FROM portfolios "
            "WHERE user_id = $1 ORDER BY id",
            user_id
        );
```

Create Portfolio

```
 pqxx::result R = W.exec_params(
            "INSERT INTO portfolios (user_id, name, cash) "
            "VALUES ($1, $2, $3) RETURNING id",
            user_id,
            name,
            initial_cash
        );
```

Update Cash

```
 W.exec_params(
            "UPDATE portfolios SET cash = $1 WHERE id = $2",
            new_cash,
            portfolio_id
        );
        W.commit();
```

Update Quantity

```
if (quantity <= 0) {
            W.exec_params(
                "DELETE FROM holdings WHERE portfolio_id = $1 AND symbol =
$2",
                portfolio_id,
                symbol
            );
        } else {
            W.exec_params(
                "INSERT INTO holdings (portfolio_id, symbol, quantity) "
                "VALUES ($1, $2, $3) "
                "ON CONFLICT (portfolio_id, symbol) "
                "DO UPDATE SET quantity = EXCLUDED.quantity",
                portfolio_id,
                symbol,
                quantity
            );
        }
```

Load Holdings for Portfolio

```
pqxx::result R = W.exec_params(
            "SELECT symbol, quantity "
            "FROM holdings "
            "WHERE portfolio_id = $1 "
            "ORDER BY symbol",
            portfolio_id
        );
```

FRIEND RELATED

Get Friends

```
pqxx::result R = W.exec_params(
            "SELECT u.id, u.username FROM users u "
            "INNER JOIN friendships f ON "
            "   (f.user1_id = $1 AND f.user2_id = u.id) OR "
            "   (f.user2_id = $1 AND f.user1_id = u.id) "
            "ORDER BY u.username",
            user_id
        );
```

Get Incoming Requests

```
pqxx::result R = W.exec_params(
```

```
            "SELECT fr.id, fr.from_user_id, fr.to_user_id, "
            "       u.username as from_username, "
            "       fr.created_at::text, "
            "       COALESCE(fr.rejected_at::text, '') as rejected_at "
            "FROM friend_requests fr "
            "INNER JOIN users u ON fr.from_user_id = u.id "
            "WHERE fr.to_user_id = $1 AND fr.rejected_at IS NULL "
            "ORDER BY fr.created_at DESC",
            user_id
        );
```

Get Outgoing Requests

```
 pqxx::result R = W.exec_params(
            "SELECT fr.id, fr.from_user_id, fr.to_user_id, "
            "       u.username as to_username, "
            "       fr.created_at::text, "
            "       COALESCE(fr.rejected_at::text, '') as rejected_at "
            "FROM friend_requests fr "
            "INNER JOIN users u ON fr.to_user_id = u.id "
            "WHERE fr.from_user_id = $1 AND fr.rejected_at IS NULL "
            "ORDER BY fr.created_at DESC",
            user_id
        );
```

Send Friend Request

```
W.exec_params(
            "INSERT INTO friend_requests (from_user_id, to_user_id) VALUES
($1, $2)",
            from_user_id, to_user_id
        );
```

Accept Friend Request

```
pqxx::result R = W.exec_params(
            "SELECT from_user_id, to_user_id FROM friend_requests WHERE id
= $1 AND rejected_at IS NULL",
            request_id
        );
W.exec_params(
            "INSERT INTO friendships (user1_id, user2_id) VALUES ($1, $2)
"
            "ON CONFLICT DO NOTHING",
            u1, u2
```

```
        );
W.exec_params(
            "DELETE FROM friend_requests WHERE id = $1",
            request_id
        );
```

Reject Friend Request

```
W.exec_params(
            "UPDATE friend_requests SET rejected_at = NOW() WHERE id =
$1",
            request_id
        );
```

Remove Friend

```
W.exec_params(
            "DELETE FROM friendships WHERE user1_id = $1 AND user2_id =
$2",
            u1, u2
        );
```

Search Users

```
pqxx::result R = W.exec_params(
            "SELECT id, username FROM users "
            "WHERE username ILIKE $1 AND id != $2 "
            "ORDER BY username LIMIT 50",
            "%" + search_text + "%",
            exclude_user_id
        );
```

STOCK RELATED

Read All from Stock symbol

```
pqxx::result R = W.exec_params(
            "SELECT timestamp, open, high, low, close, volume "
            "FROM Stocks "
            "WHERE symbol = $1 "
            "ORDER BY timestamp ASC",
            symbol
        );
```

Get all symbols

```
pqxx::result R = W.exec(
```

```
            "SELECT DISTINCT symbol FROM Stocks ORDER BY symbol ASC"
        );
```

Find a symbol

```
pqxx::result R = W.exec_params(
            "SELECT DISTINCT symbol FROM Stocks WHERE symbol ILIKE $1
ORDER BY symbol ASC",
            std::string("%") + target + std::string("%")
        );
```

Get MA statistics

```
pqxx::result R = W.exec_params(
            "SELECT "
            "  (SELECT AVG(close) FROM (SELECT close FROM Stocks WHERE
symbol=$1 ORDER BY timestamp DESC LIMIT 5) t) as ma5, "
            "  (SELECT AVG(close) FROM (SELECT close FROM Stocks WHERE
symbol=$1 ORDER BY timestamp DESC LIMIT 10) t) as ma10, "
            "  (SELECT AVG(close) FROM (SELECT close FROM Stocks WHERE
symbol=$1 ORDER BY timestamp DESC LIMIT 30) t) as ma30, "
            "  (SELECT AVG(close) FROM (SELECT close FROM Stocks WHERE
symbol=$1 ORDER BY timestamp DESC LIMIT 60) t) as ma60, "
            "  (SELECT close FROM Stocks WHERE symbol=$1 ORDER BY
timestamp DESC LIMIT 1) as current_price",
            symbol
        );
```

Get general stats

```
 pqxx::result R = W.exec_params(
            "SELECT "
            "  AVG(close) as mean, "
            "  STDDEV(close) as stddev, "
            "  CASE WHEN AVG(close) > 0 THEN STDDEV(close) / AVG(close)
ELSE 0 END as cov "
            "FROM Stocks "
            "WHERE symbol = $1 "
            "  AND timestamp >= $2 AND timestamp <= $3",
            symbol, start_date, end_date
        );
```

Get beta `Covariance(stock, market) / Variance(market)`

```
pqxx::result R = W.exec_params(
            "WITH stock_returns AS ("
            "  SELECT timestamp, "
```

```
            "    (close - LAG(close) OVER (ORDER BY timestamp)) /
LAG(close) OVER (ORDER BY timestamp) as return "
            "  FROM Stocks "
            "  WHERE symbol = $1 AND timestamp >= $2 AND timestamp <= $3"
            "), "
            "market_returns AS ("
            "  SELECT timestamp, "
            "    AVG((close - prev_close) / prev_close) as market_return "
            "  FROM ("
            "    SELECT timestamp, close, LAG(close) OVER (PARTITION BY
symbol ORDER BY timestamp) as prev_close "
            "    FROM Stocks "
            "    WHERE timestamp >= $2 AND timestamp <= $3"
            "  ) t "
            "  WHERE prev_close IS NOT NULL AND prev_close > 0 "
            "  GROUP BY timestamp"
            ") "
            "SELECT "
            "  CASE WHEN VAR_POP(m.market_return) > 0 "
            "    THEN COVAR_POP(s.return, m.market_return) /
VAR_POP(m.market_return) "
            "    ELSE 0 END as beta "
            "FROM stock_returns s "
            "JOIN market_returns m ON s.timestamp = m.timestamp "
            "WHERE s.return IS NOT NULL",
            symbol, start_date, end_date
        );
```

Get correlation

```
pqxx::result R = W.exec_params(
            "SELECT "
            "  COVAR_POP(s1.close, s2.close) as covariance, "
            "  CORR(s1.close, s2.close) as correlation "
            "FROM Stocks s1 "
            "JOIN Stocks s2 ON s1.timestamp = s2.timestamp "
            "WHERE s1.symbol = $1 AND s2.symbol = $2 "
            "  AND s1.timestamp >= $3 AND s1.timestamp <= $4",
            symbol1, symbol2, start_date, end_date
        );
```

Stockdb add price

```
W.exec_params(
            "INSERT INTO Stocks (symbol, timestamp, open, close, high,
low, volume) "
            "VALUES ($1, $2, $3, $4, $5, $6, $7) "
            "ON CONFLICT (symbol, timestamp) DO UPDATE SET "
            "open = EXCLUDED.open, close = EXCLUDED.close, "
            "high = EXCLUDED.high, low = EXCLUDED.low, volume =
EXCLUDED.volume",
            symbol, date, open, close, high, low, volume
        );
```

STOCKLIST RELATED

Get all stocklists

```
pqxx::result R = W.exec_params(
            "SELECT DISTINCT sl.id, sl.name, sl.user_id, sl.visibility, "
            "u.username AS owner_username "
            "FROM stocklists sl "
            "JOIN users u ON sl.user_id = u.id "
            "LEFT JOIN stocklist_to_user stu ON sl.id = stu.stocklist_id "
            "WHERE sl.user_id = $1 OR stu.user_id = $1 OR sl.visibility =
0 "
            "ORDER BY sl.id",
            user_id
        );
```

Get stocklist description

```
pqxx::result R_desc = W.exec_params(
                "SELECT description FROM stocklist_descriptions WHERE
stocklist_id = $1",
                (*stocklists)[i]->stocklist_id
            );
```

Get stocklist shared users

```
pqxx::result R_shared = W.exec_params(
                "SELECT u.username FROM stocklist_to_user stu "
                "JOIN users u ON stu.user_id = u.id "
                "WHERE stu.stocklist_id = $1",
                (*stocklists)[i]->stocklist_id
            );
```

Get stocks in stocklist

```
pqxx::result R = W.exec_params(
            "SELECT hs.symbol, hs.quantity, hs.stocklist_id "
```

```
            "FROM stocklist_holdings hs "
            "WHERE hs.stocklist_id = $1",
            stocklist->stocklist_id
        );
```

Get reviews

```
pqxx::result R2 = W2.exec_params(
            "SELECT sr.reviewer_id, sr.review, u.username "
            "FROM stocklist_reviews sr "
            "JOIN users u ON sr.reviewer_id = u.id "
            "WHERE sr.stocklist_id = $1",
            stocklist->stocklist_id
        );
```

Create a stocklist

```
pqxx::result R = W.exec_params(
            "INSERT INTO stocklists (name, user_id, visibility) "
            "VALUES ($1, $2, $3) RETURNING id",
            stocklist->name,
            user_id,
            (int)(stocklist->visibility)
        );
```

Modify a stock in stocklist

```
if(new_quantity<=0){
            //remove stock from stocklist
            W.exec_params(
                "DELETE FROM stocklist_holdings WHERE stocklist_id = $1
AND symbol = $2",
                stocklist->stocklist_id,
                symbol
            );
            print_to_debug("Removed stock from stocklist");
        }else{
            //insert and resolve conflict by updating quantity
            W.exec_params(
                "INSERT INTO stocklist_holdings (stocklist_id, symbol,
quantity) "
                "VALUES ($1, $2, $3) "
                "ON CONFLICT (stocklist_id, symbol) DO UPDATE SET quantity
= EXCLUDED.quantity",
                stocklist->stocklist_id,
                symbol,
```

```
                new_quantity
            );
            print_to_debug("Upserted stock into stocklist");
        }
```

Modify shared users

```
W.exec_params(
                            "INSERT INTO stocklist_to_user (stocklist_id,
user_id) VALUES ($1, $2)",
                            stocklist->stocklist_id,
                            to_user_id
                        );
```

Modify description

```
W.exec_params(
            "UPDATE stocklist_descriptions SET description = $1 WHERE
stocklist_id = $2",
            stocklist->description,
            stocklist->stocklist_id
        );
```

Delete a stocklist

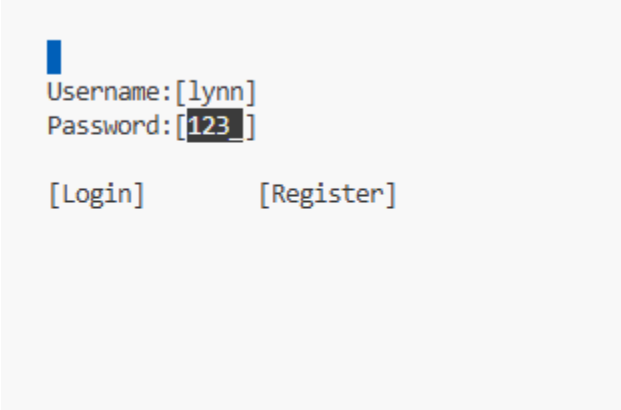
```
W.exec_params(
            "DELETE FROM stocklists WHERE id = $1",
            stocklist->stocklist_id
        );
```

Modify a review

```
W.exec_params(
            "INSERT INTO stocklist_reviews (stocklist_id, reviewer_id,
review) "
            "VALUES ($1, $2, $3) "
            "ON CONFLICT (stocklist_id, reviewer_id) "
            "DO UPDATE SET review = EXCLUDED.review",
            s->stocklist_id,
            reviewer_id,
            review->review_text
        );
```

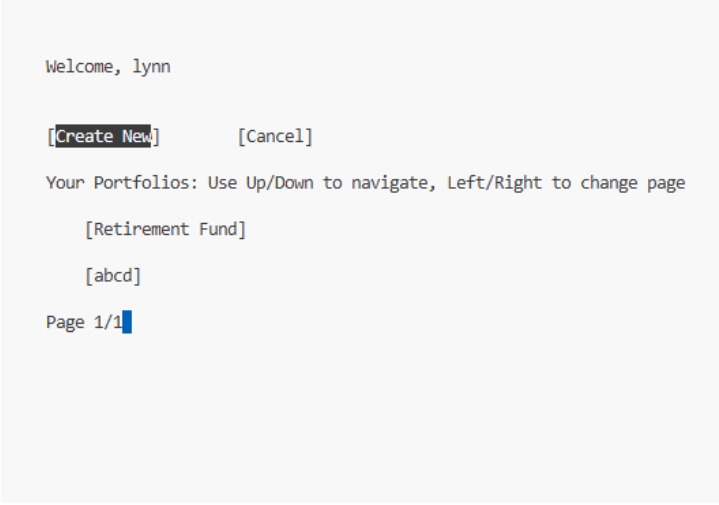
UI implementation

Login Page



Main Menu

Portfolio Menu



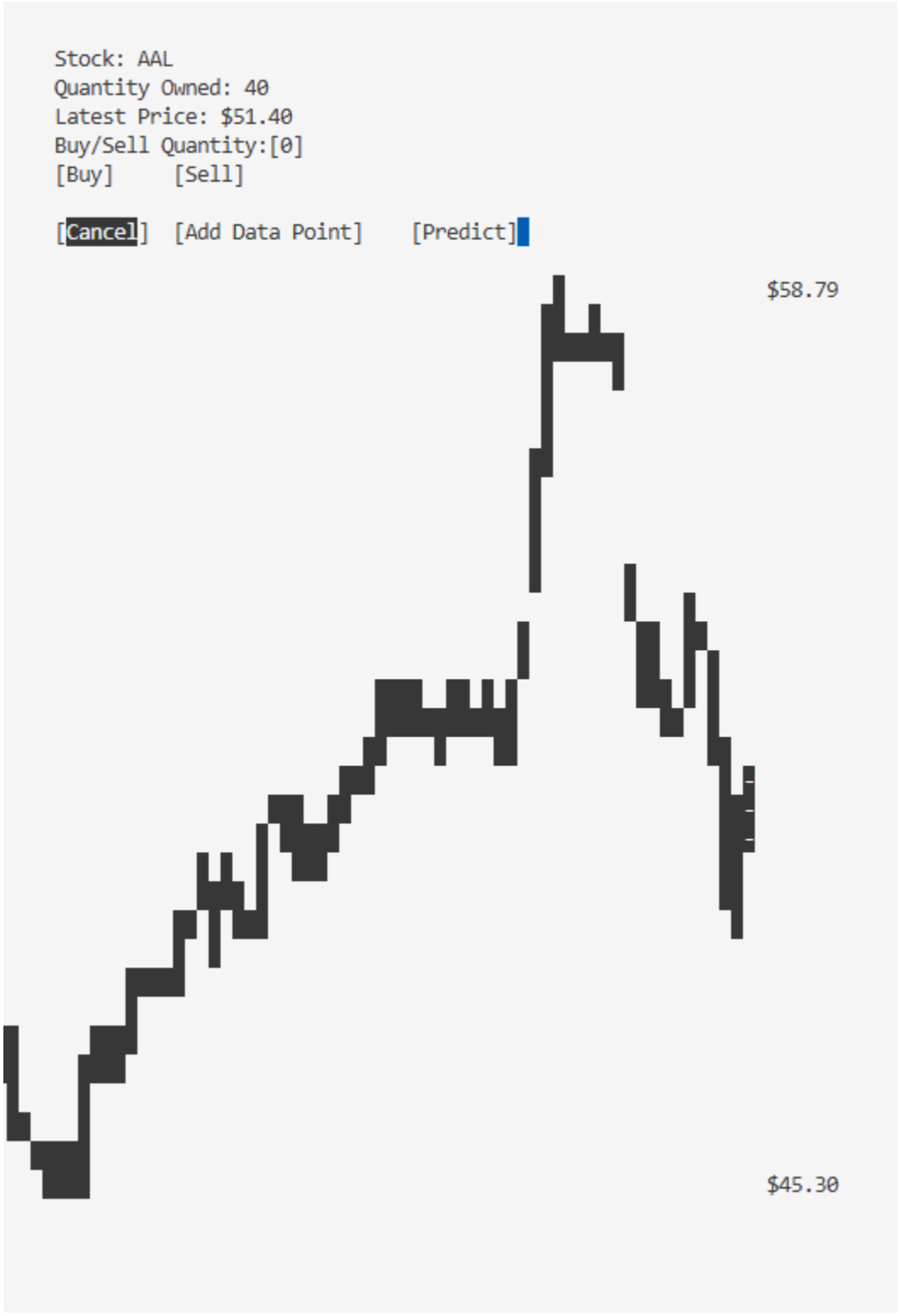
Portfolio Management

```
Portfolio: Retirement Fund
Cash Available: $6809              Total Stock Value: $3192

W/D Amount:[0]
[Withdraw][Deposit]

[Purchase New Stock]     [Stats]
[Cancel]

Your Stocks: Use Up/Down to navigate, Left/Right to change page

    [AAL (40 shares, 51.40 USD each)]

    [ABBV (10 shares, 113.62 USD each)]

Page 1/1
```

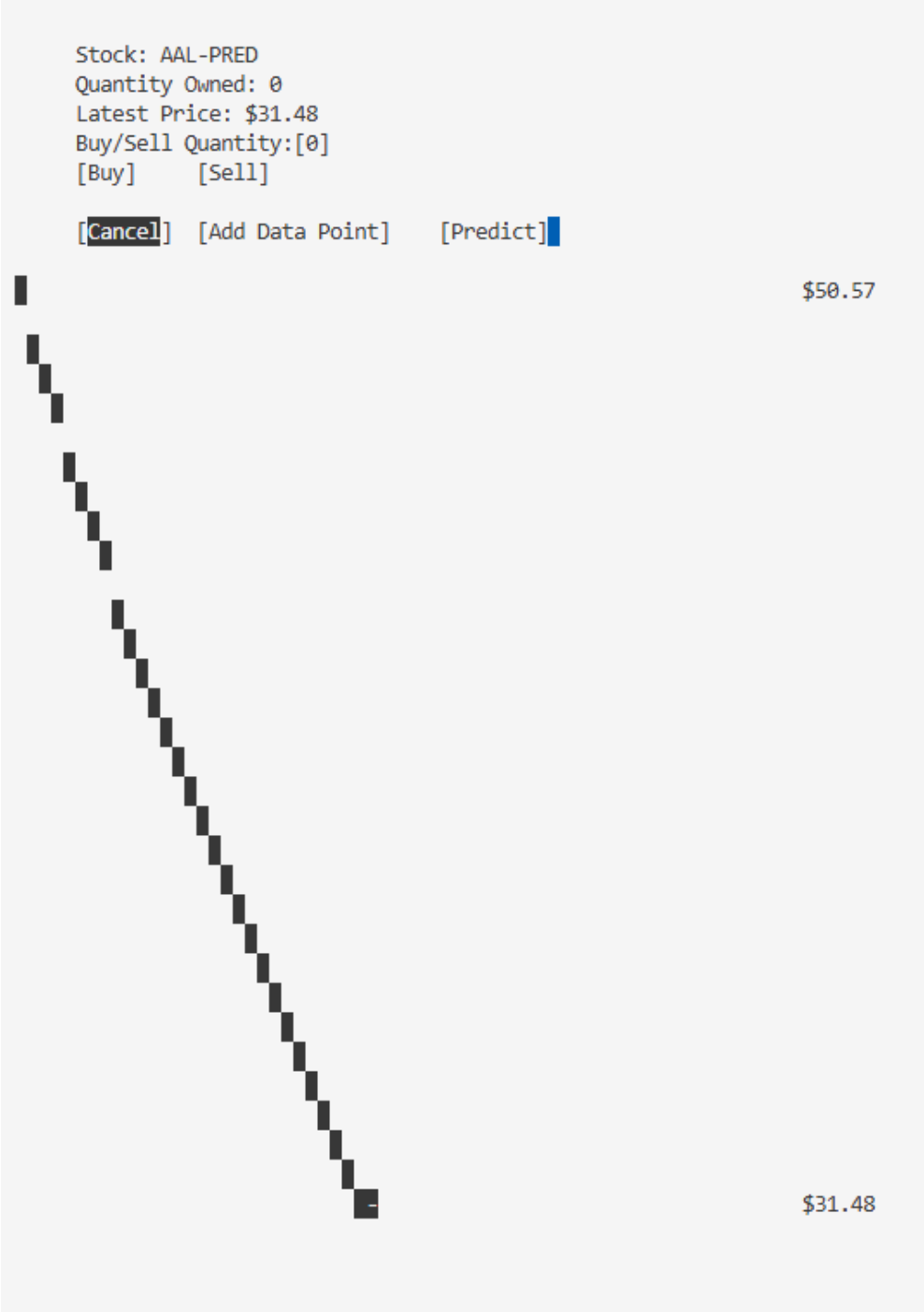
Portfolio Stats

```
Portfolio Statistics: Retirement Fund
Period: 2010-12-01 to 2018-02-07
[Back]
=== Stock Statistics (COV & Beta) ===
Symbol           Mean     StdDev        COV       Beta
-----------------------------------------------------
AAL             38.44      11.09     0.2885     1.4404
ABBV            60.86      13.73     0.2256     1.0237

=== Correlation Matrix ===
                 AAL      ABBV
AAL           1.0000    0.7231
ABBV          0.7231    1.0000
```

Stock Display Menu

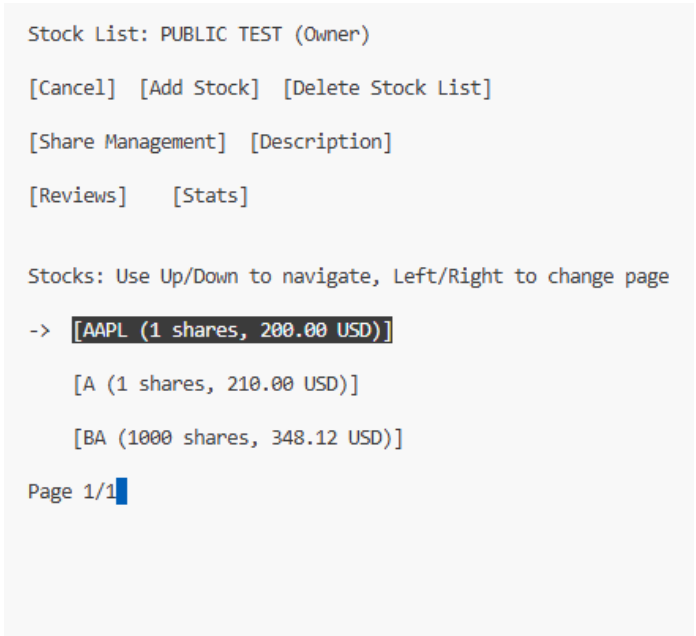


Predictions



Stocklist Menu

Stocklist Menu (selected)



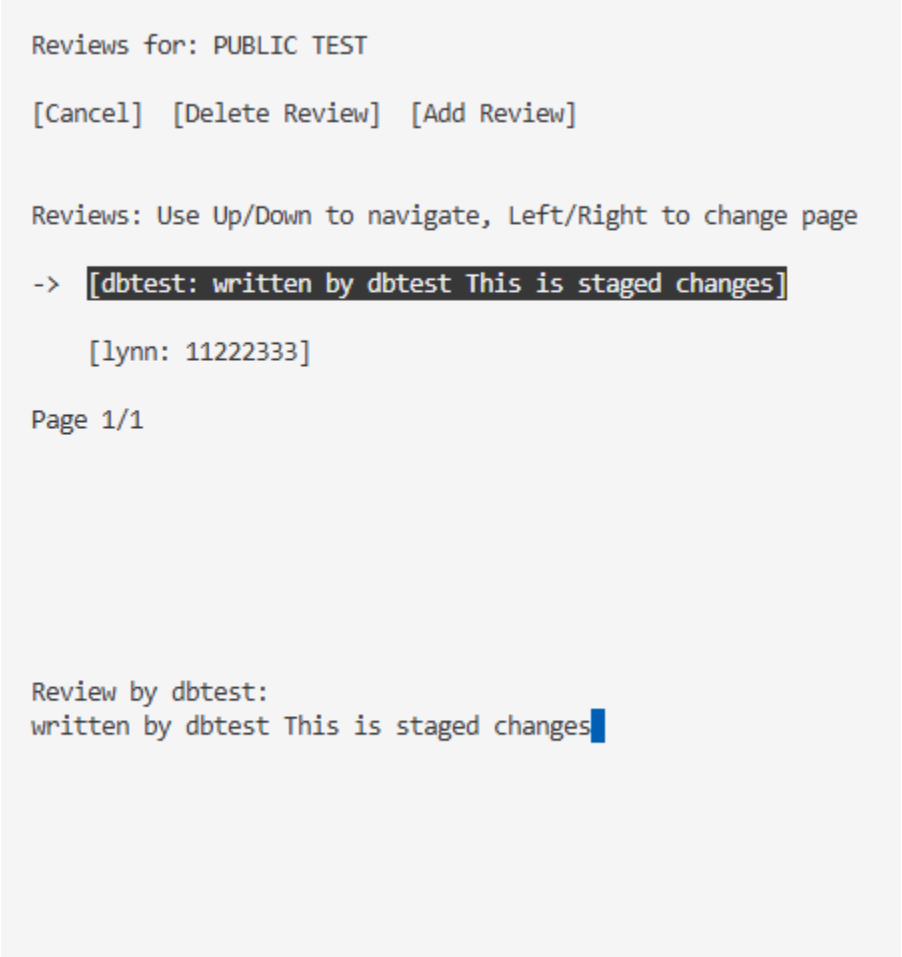
Stocklist Stats

```
Stock List Statistics: PUBLIC TEST
Period: 2000-12-01 to 2030-12-02
[Back]
=== Stock Statistics (COV & Beta) ===
Symbol           Mean     StdDev        COV       Beta
-----------------------------------------------------
AAPL           109.12      30.63     0.2807     0.6509
A               49.45      11.12     0.2248     1.0038
BA             149.91      49.78     0.3321     1.0158

=== Correlation Matrix ===
                AAPL         A        BA
AAPL          1.0000    0.4004    0.8640
A             0.4004    1.0000    0.6867
BA            0.8640    0.6867    1.0000
```

Add stock menu

```
lynn - Stock Search

Search:[ ]

[Search]      [Cancel]

Results: Use Up/Down to navigate, Left/Right to change page

    [A]

    [AAL]

    [AAP]

    [AAPL]

    [ABBV]

Page 1/101
```

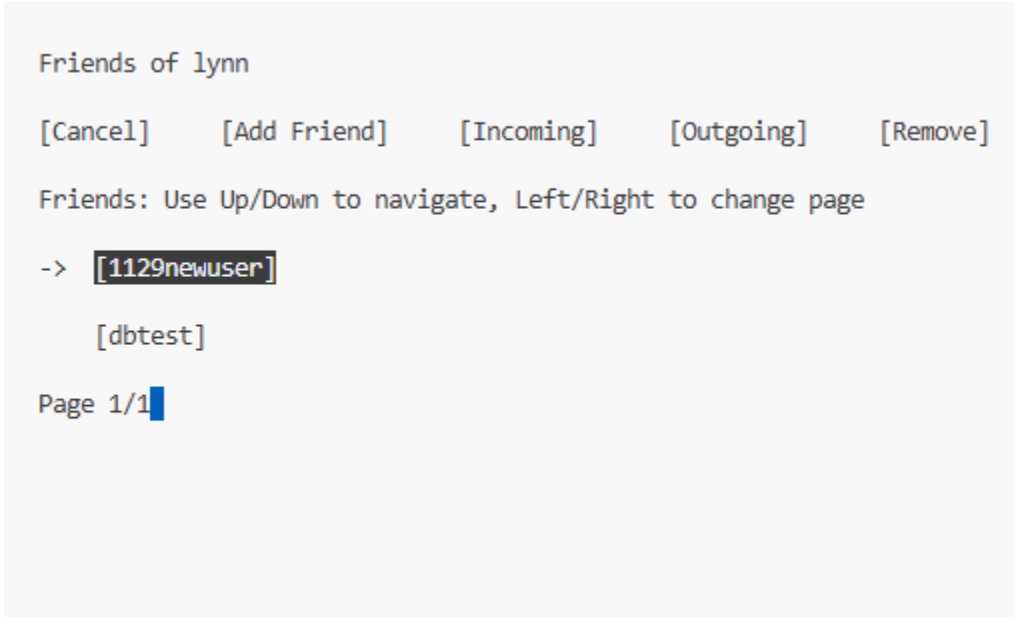
Review menu



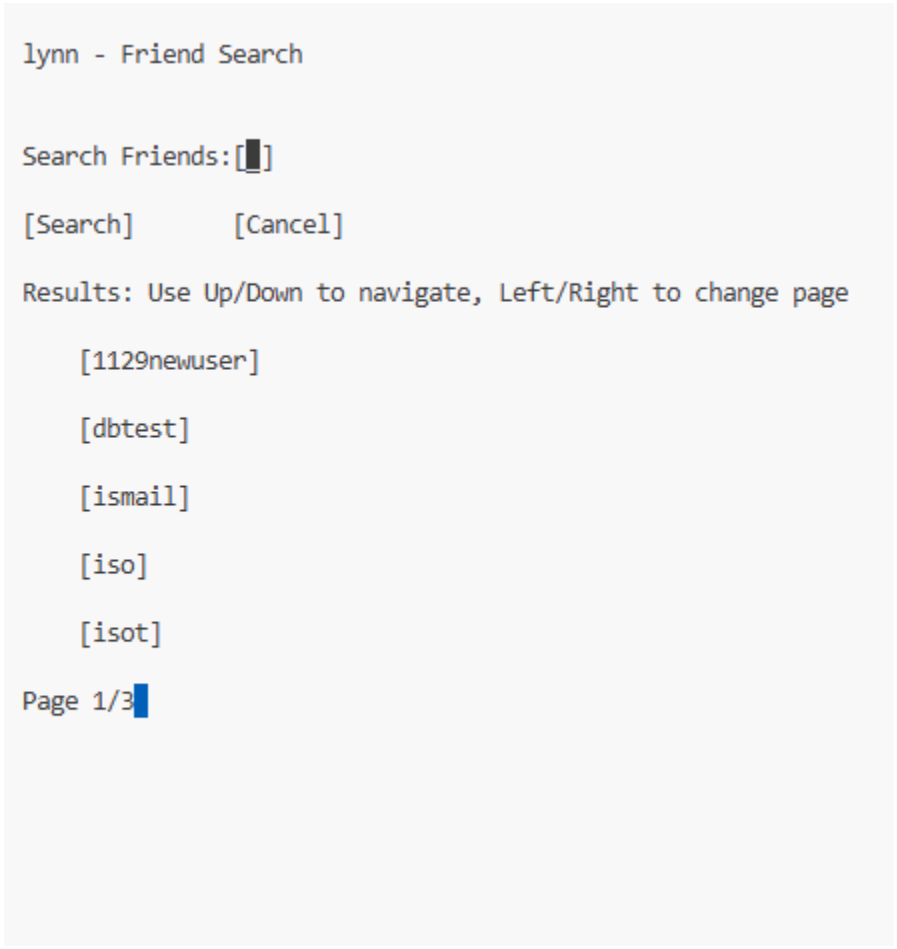
Reading

Editing

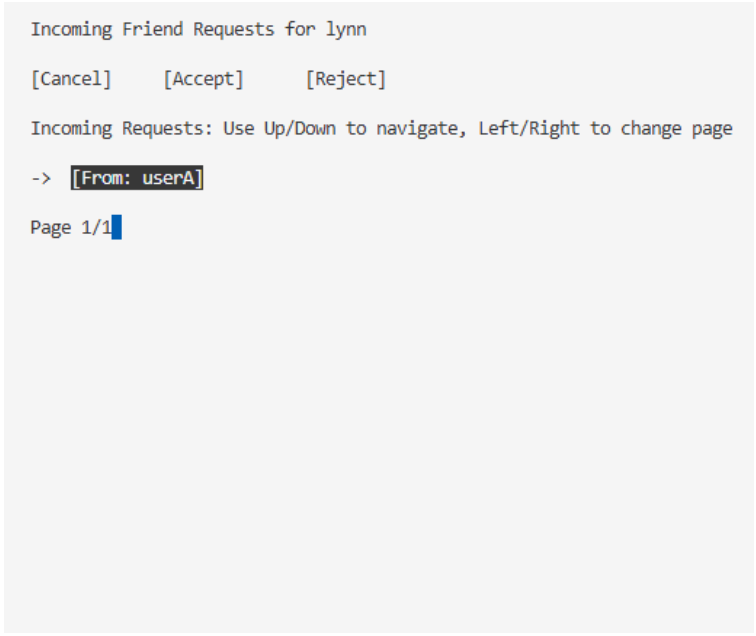
Friend Menu



Friend Search



View Pending requests



Pending Outgoing

Final notes to test our code:
Database name: mydb
It must pre-load table of the csv of the stocks according to the env setup document.
Run ./main, other tables will be generated automatically.