**JDBC Project: Stock Market Exchange**

**Abhinav Gupta** (120050030) **Ainesh Pandey** (14V051001) **Aman Gour** (120050030)

Overview

Our simple financial product VSE provides an elegant, user-friendly interface for the user which runs on a robust back-end implementation designed to handle all the intricacies of real world stock market. For our JDBC project, we simulated a stock market exchange website modeled after two kinds of users: independent enterprising individuals and companies offering stock.

Implementation

Our application features robust code with multiple concurrent user access. This application is divided into two major components: the user-side and the company-side. For the backend, we utilized the HttpServlet superclass to connect to a local database on our servers.

**User Side**

The user side interface works on the following servlets, custom classes, and jsps

* UserLogin: This servlet deals with all the functionality related to user login and registration
  + Login:Provides a fully secure login functionality
  + Register: Provides a functionality for new user to make an account on VSE and then redirects to the Login page
  + Search: This feature uses an asynchronous dynamic refresh feature that provides a quick access to the database to fetch only new information
* Transactions: A servlet to provides functionality of exchanges of shares over the exchange. The stock exchange involves frequent transactions and so we have optimised this servlet to meet the expectation
  + MarketBuyOrders: Market buy orders are used by shareholders to quickly buy shares from the exchange at market price without specifically Bid Price.
  + LimitBuyOrders: Limit orders adds the functionality of price specification by the user. These orders are stored in buffers for efficient access.
  + MarketSellOrders: Market buy orders are used by shareholders to quickly buy shares from the exchange at market price without specifically Ask Price.
  + LimitSellOrders: Limit orders adds the functionality of price specification by the user. These orders are stored in buffers for efficient access.

**Company Side**

Most of the essential stock market features are offered on the user side, while we limit the company’s operations to

* launching IPOs
* paying dividends to shareholders

We made the code robust to handle some errors on the human users part.

* CompanyLogin (servlet): This servlet deals with all the functionality related to company login and registration
  + CompanyLogin:Provides a fully secure login functionality, along with the company’s stock information and transaction records for the homepage
  + CompanyRegister: Provides a functionality for new user to make an account on VSE and then redirects to the CompanyLogin page
* LaunchIPO (servlet): This servlet launches a company’s stock by a) creating a new stock in our database with the company’s stock symbol and b) creating a sellOrder by the user ‘admin’ containing all of the stocks of the company.
* PayDividends (servlet): This servlet pays a flat dividend/share to all of the shareholders of the company’s stock by a) selecting all of the tuples from the table ownership that have the company’s stock symbol and b) updating the users table for all of the users according to how many shares they own.
* TransRecord (custom class): This class acts as a struct to store tuples from transactions table into a vector of TransRecords.
* CompanyStock (custom class): This class acts as a struct to store the information about the company’s stock.

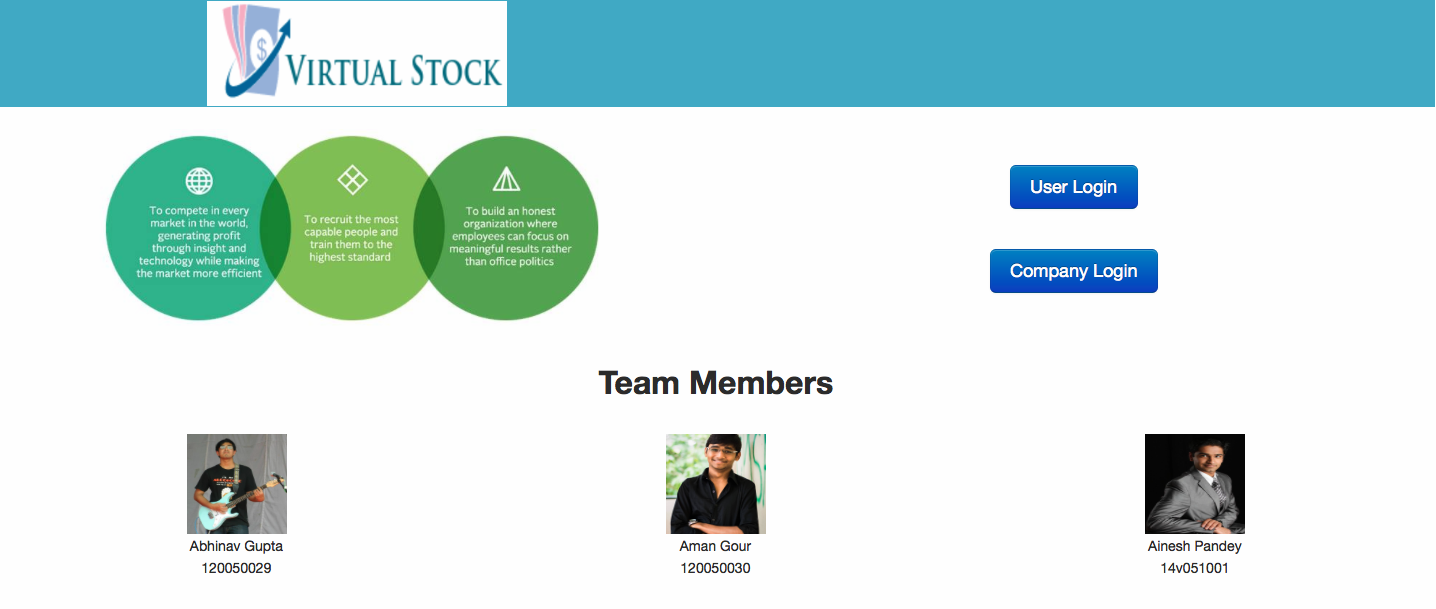
**Admin Functionality**

We intended to provide admin functionality where the admin can keep a check on all the activities of the stock exchange and also have the access to the database and so if there is some malicious activity by the user or company appropriate actions can be taken.

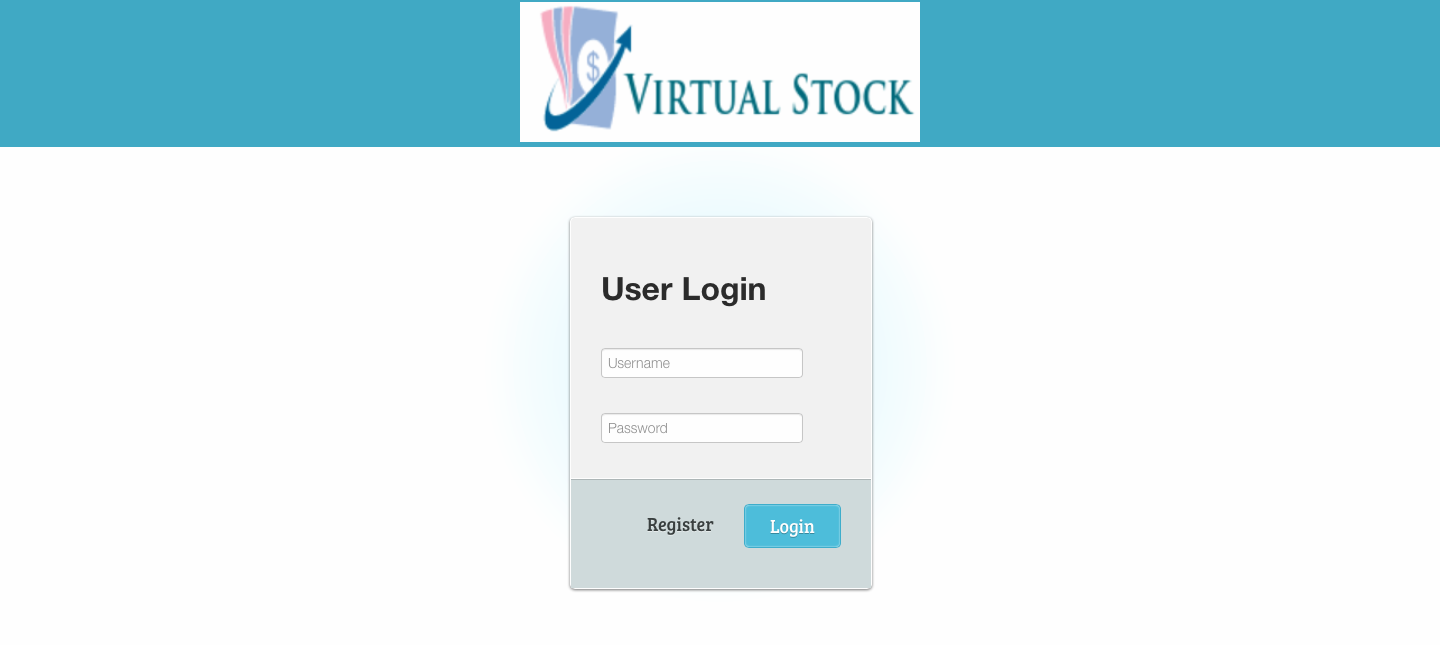
Interfaces

Application provides an elegant interfaces embedded with all the informations a user needs to make intelligent decision like plots, share movement and transaction details. Our implementation required the implementation of multiple user interfaces, which we generated through the use of HTML, JSP, CSS, AJAX and other useful libraries of java.

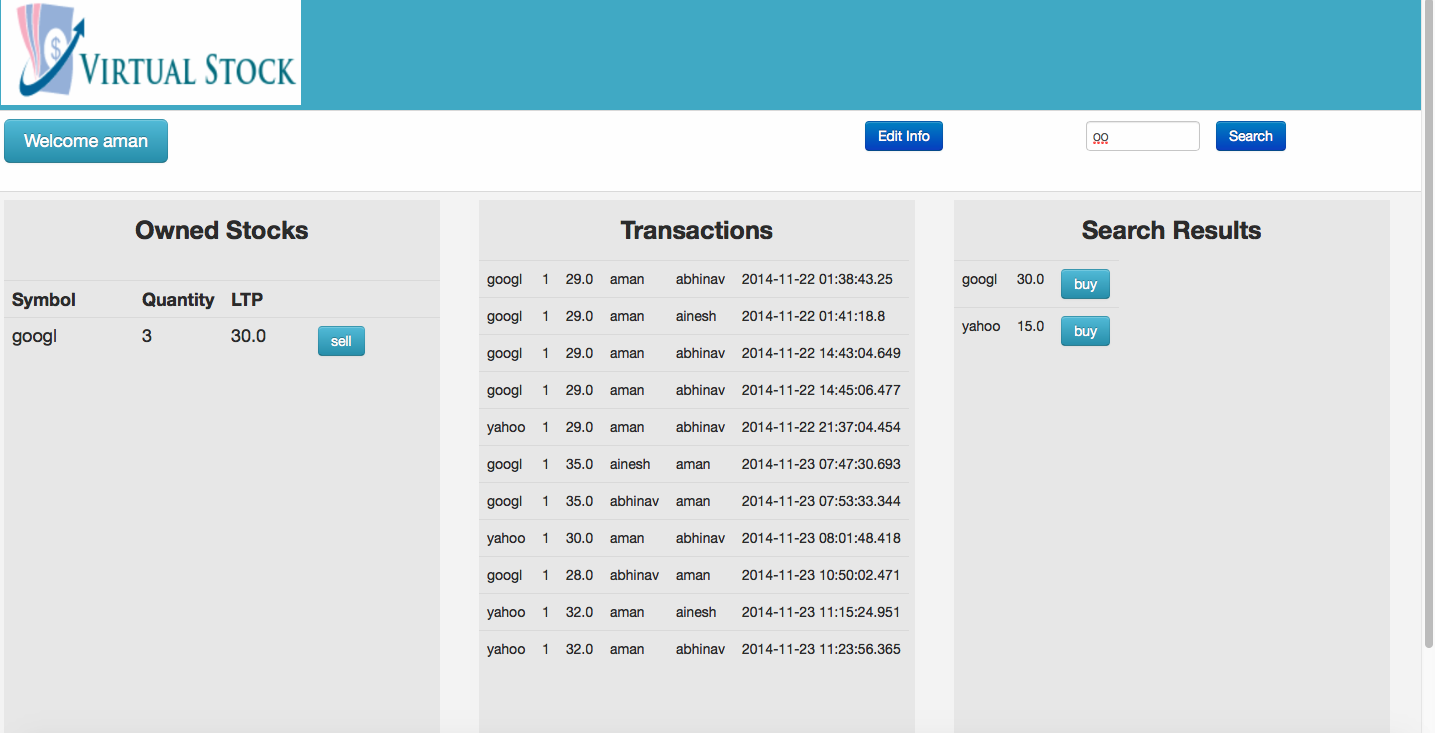
HOME



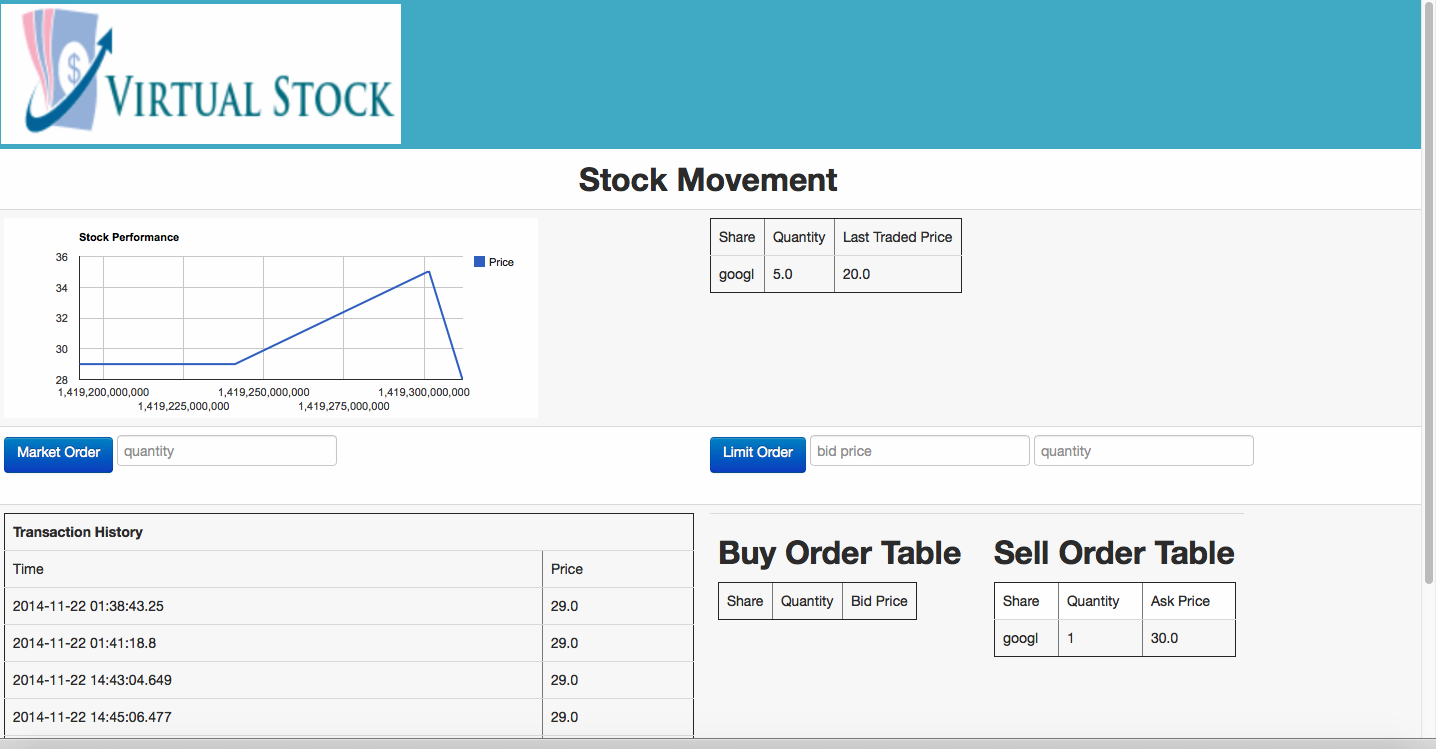
USER/COMPANY LOGIN



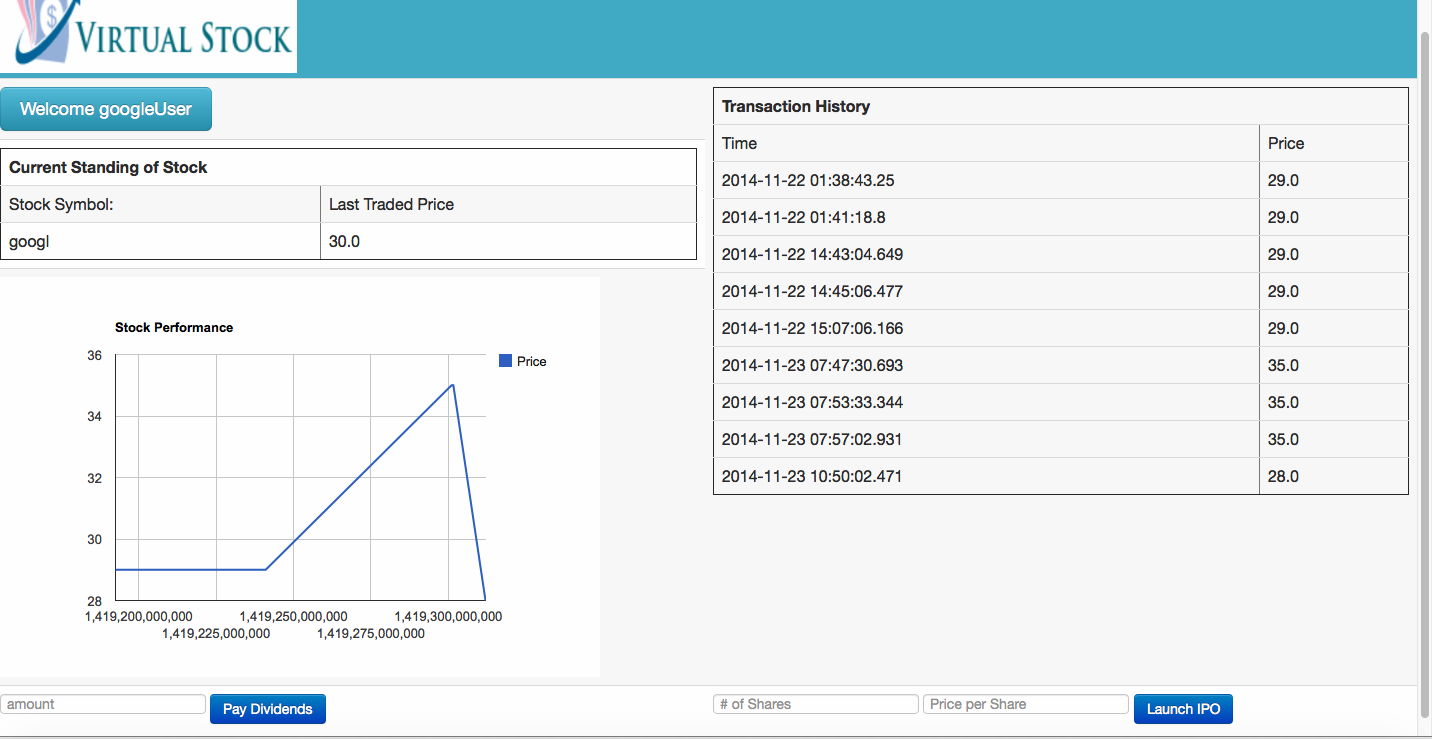
USER HOME



ORDER FRAME



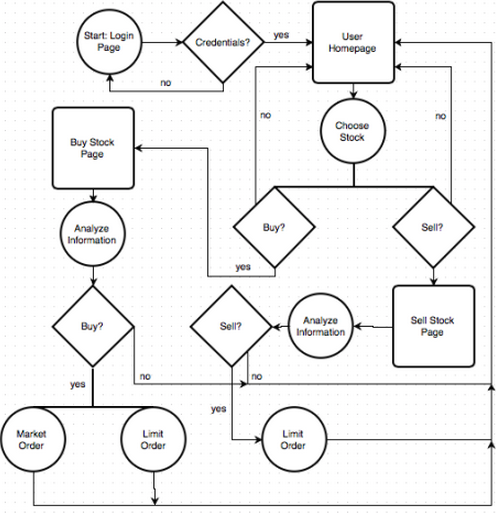
COMPANY HOME



Flow Diagram

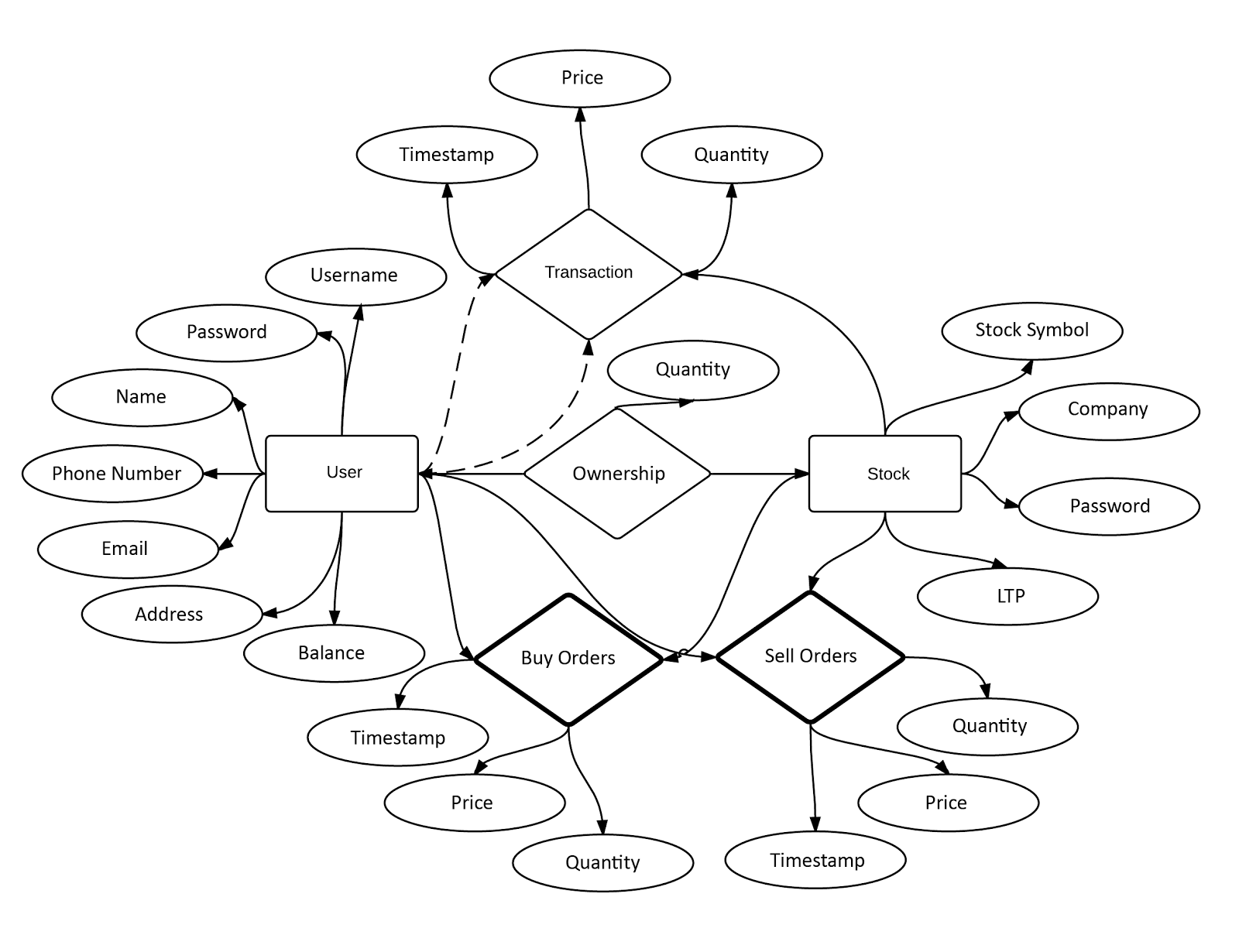
**User Side-** After providing the pertinent personal information to register for the website, users have the capability of logging in with a username and password. There, they can check the status of the stocks they possess, view their transaction history, and search for stocks in the market. The searching mechanism provides users with a graph depicting recent activity of the stock and available buy/sell orders.

**Company Side-** Companies as well provide pertinent information to register as a company on the website, after which they can login to our portal with their username and password. Once signed up, companies can choose when to launch their IPO, upon which a new stock will be added to the records and a sell order with their newly created stock will be added to the stock market. They also have the ability to pay dividends to their users.



Roots - ER, Schema Diagram

The whole application was based on the ER diagram which was designed to reduce the data redundancy and to provides constraints to provide data consistency.

**ER Diagram**

Schema

We modified the 3NF schema to provide a more intuitive schema that includes a separate table for companies earlier we just had the stock launched by company. This additional table helped us identity each stock as belonging to a company and also eased the task for dividends and IPO’s.

The following diagram shows the changes in red box. We used normalization to eliminate data redundancy and undesirable characteristics.

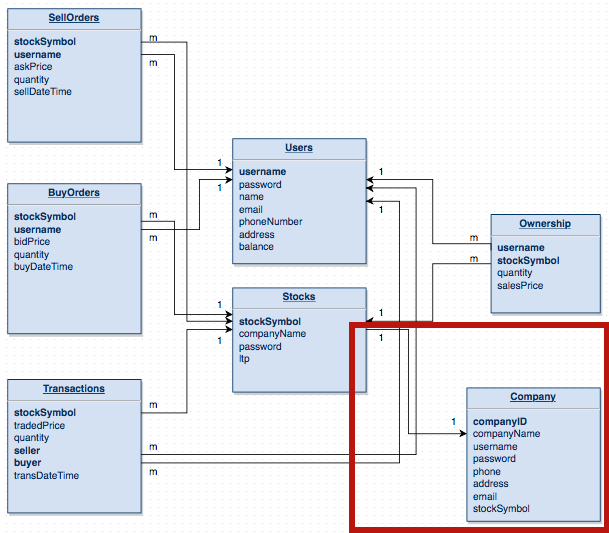
Normalization of our database helped us to serve two main purposes :

1. Eliminating redundant data
2. Ensuring data dependencies make sense i.e. data is logically stored.

We started out with 1-NF and then went higher in the hierarchy to remove the redundancy in data and to integrate the data well.

Current version of database design implements 3NF where we have removed the transitive functional dependency between the *Transaction* table and the *Users* and *Stocks* table.

MODIFIED SCHEMA DIAGRAM



References

* [**http://www.postgresql.org/**](http://www.postgresql.org/)
* [**http://www.bseindia.com/**](http://www.bseindia.com/)
* [**http://www.nseindia.com/**](http://www.nseindia.com/)
* [**http://en.wikipedia.org/wiki/Stock\_market\_simulator**](http://en.wikipedia.org/wiki/Stock_market_simulator)
* [**http://en.wikipedia.org/wiki/World\_Stock\_Exchange**](http://en.wikipedia.org/wiki/World_Stock_Exchange)
* [**http://www.w3schools.com/ajax/**](http://www.w3schools.com/ajax/)
* [**http://www.tutorialspoint.com/jdbc/**](http://www.tutorialspoint.com/jdbc/)