Grow High - Stock Order Management System (SOMS)

Project Report

Institution: *Department of Computer Science, Georgia State University*

Team Members: *Nagamedha Sakhamuri - 002828574 ; Ttanvi Tummapudi - 002843956*

INTRODUCTION

Application Overview: *Grow High* is a Stock Order Management System (SOMS) designed to facilitate seamless and efficient stock trading. The platform enables custodians to manage clients and their trading portfolios effectively while automating stock transactions. It provides functionalities for buying, selling, and maintaining trade histories, with features such as real-time updates, automated email notifications, and advanced database handling.

MOTIVATION

Managing stock trading processes manually can lead to inefficiencies, errors, and a lack of transparency. This inspired the creation of SOMS, which focuses on leveraging database-driven operations to simplify trading workflows while maintaining data consistency. Unlike traditional systems, Grow High integrates advanced features such as stored procedures for transaction handling and automated notifications for enhanced user experience.

Key Components

- **Frontend**: Built using AngularJS for responsive and interactive interfaces.
- **Backend**: Implemented using Spring Boot for robust API development.
- **Database**: Powered by MariaDB to handle complex transactions and large datasets efficiently.

DATABASE DETAILS

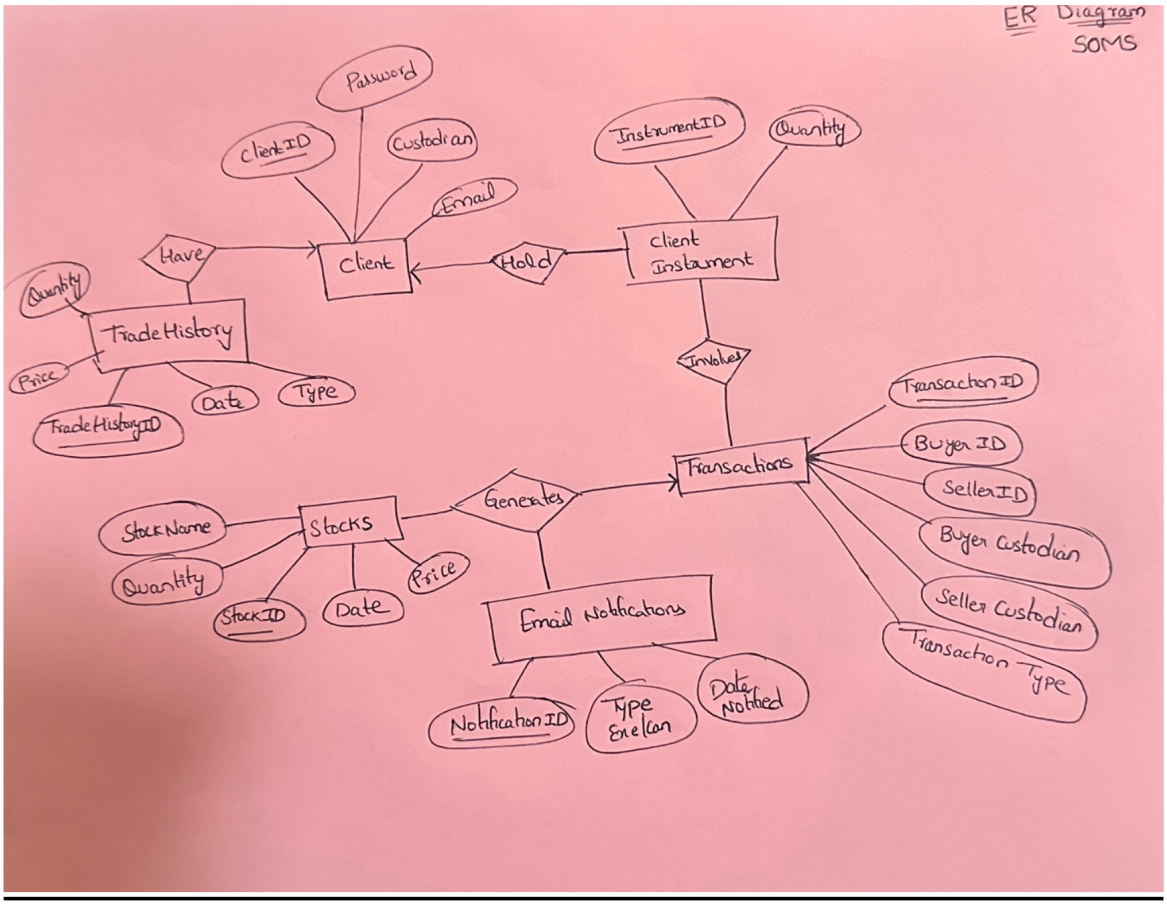
Database Design and Models

- **E-R Model**

The database is designed to support the **Stock Order Management System (SOMS)** with the following entities:

1. **Client**: Represents stock trading clients managed by custodians.
2. **Custodian**: Represents stockbrokers managing clients and their trades.

3. **Instrument**: Represents stocks available for trading, with attributes like face value and quantity.
4. **Transactions**: Tracks executed trades between buyers and sellers.
5. **BuyStocks and SellStocks**: Handle ongoing buy and sell requests.
6. **ClientInstrument**: Represents the relationship between clients and the stocks they hold.



- **Relationships**:
  - A **Custodian** manages multiple **Clients**.
  - A **Client** holds multiple **Instruments** through the **ClientInstrument** table.
  - A **Transaction** links buyers, sellers, and instruments.
- **Relational Model**
  1. **Client Table**:

- Attributes: clientid (PK), clientname, transactionlimit, custodianid (FK).
- Stores client details and their associated custodian.

2. **Custodian Table**:
   - Attributes: custodianid (PK), custodianname, password.
   - Maintains custodian details.
3. **Instrument Table**:
   - Attributes: instrumentid (PK), instrumentname, facevalue, expirydate, quantity.
   - Represents stocks available for trading.
4. **ClientInstrument Table**:
   - Attributes: id (PK), clientid (FK), instrumentid (FK), quantity.
   - Tracks stocks held by each client.
5. **Transactions Table**:
   - Attributes: transactionid (PK), buyer_client (FK), seller_client (FK), buyer_custodian (FK), seller_custodian (FK), instrumentid (FK), transfer_date, quantity, price, total_amount.
   - Records executed trades.
6. **BuyStocks Table**:
   - Attributes: buystock_id (PK), clientid (FK), instrumentid (FK), quantity, price, buy_date.
   - Tracks active buy requests.
7. **SellStocks Table**:
   - Attributes: sellstock_id (PK), clientid (FK), instrumentid (FK), quantity, price, sell_date.
   - Tracks active sell requests.

- **Normalization**
  - **BCNF Compliance**: All tables adhere to Boyce-Codd Normal Form (BCNF), ensuring no redundancy or partial dependencies.

- **Functional Dependencies**: Each table is structured such that all attributes are functionally dependent on the primary key.

- **Constraints**

1. **Primary Keys**:
    - Uniquely identify rows in each table (e.g., transactionid in Transactions table).
2. **Foreign Keys**:
    - Maintain relationships between entities (e.g., clientid in ClientInstrument references Client table).
3. **Data Type Constraints**:
    - Ensure valid data entries (e.g., quantity is integer, price is double).
4. **Auto-Increment**:
    - Used for IDs in BuyStocks, SellStocks, and Transactions tables for unique identification.

```
-- Dumping structure for table oms.transactions
CREATE TABLE IF NOT EXISTS `transactions` (
  `transactionid` int(20) NOT NULL,
  `buyer_client` varchar(100) NOT NULL,
  `seller_client` varchar(100) NOT NULL,
  `buyer_custodian` varchar(100) NOT NULL,
  `seller_custodian` varchar(100) NOT NULL,
  `instrumentid` varchar(50) NOT NULL,
  `transfer_date` datetime NOT NULL,
  `quantity` int(11) NOT NULL,
  `price` double NOT NULL,
  `total_amount` double NOT NULL,
  PRIMARY KEY (`transactionid`),
  KEY `transactions_ibfk_1` (`buyer_client`),
  KEY `transactions_ibfk_2` (`seller_client`),
  KEY `transactions_ibfk_3` (`buyer_custodian`),
  KEY `transactions_ibfk_4` (`seller_custodian`),
  KEY `transactions_ibfk_5` (`instrumentid`),
  CONSTRAINT `transactions_ibfk_1` FOREIGN KEY (`buyer_client`) REFERENCES `client` (`clientid`),
  CONSTRAINT `transactions_ibfk_2` FOREIGN KEY (`seller_client`) REFERENCES `client` (`clientid`),
  CONSTRAINT `transactions_ibfk_3` FOREIGN KEY (`buyer_custodian`) REFERENCES `custodian` (`custodianid`),
  CONSTRAINT `transactions_ibfk_4` FOREIGN KEY (`seller_custodian`) REFERENCES `custodian` (`custodianid`),
  CONSTRAINT `transactions_ibfk_5` FOREIGN KEY (`instrumentid`) REFERENCES `instrument` (`instrumentid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
-- Dumping structure for table oms.custodian
CREATE TABLE IF NOT EXISTS `custodian` (
  `custodianid` varchar(100) NOT NULL,
  `custodianname` varchar(100) NOT NULL,
  `password` varchar(100) NOT NULL,
  PRIMARY KEY (`custodianid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
-- Dumping structure for table oms.sellstocks
CREATE TABLE IF NOT EXISTS `sellstocks` (
  `sellstock_id` int(100) NOT NULL AUTO_INCREMENT,
  `clientid` varchar(100) NOT NULL,
  `instrumentid` varchar(50) NOT NULL,
  `quantity` int(11) NOT NULL,
  `price` double NOT NULL,
  `sell_date` datetime NOT NULL,
  PRIMARY KEY (`sellstock_id`),
  KEY `sell_stocks_ibfk_1` (`clientid`),
  KEY `sell_stocks_ibfk_2` (`instrumentid`),
  CONSTRAINT `sell_stocks_ibfk_1` FOREIGN KEY (`clientid`) REFERENCES `client` (`clientid`),
  CONSTRAINT `sell_stocks_ibfk_2` FOREIGN KEY (`instrumentid`) REFERENCES `instrument` (`instrumentid`)
) ENGINE=InnoDB AUTO_INCREMENT=17 DEFAULT CHARSET=latin1;
```

```
-- Dumping structure for table oms.clientinstrument
CREATE TABLE IF NOT EXISTS `clientinstrument` (
  `clientid` varchar(100) NOT NULL,
  `instrumentid` varchar(50) NOT NULL,
  `quantity` int(11) NOT NULL,
  `id` int(10) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`),
  KEY `clientinstrument_ibfk_1` (`clientid`),
  KEY `clientinstrument_ibfk_2` (`instrumentid`),
  CONSTRAINT `clientinstrument_ibfk_1` FOREIGN KEY (`clientid`) REFERENCES `client` (`clientid`),
  CONSTRAINT `clientinstrument_ibfk_2` FOREIGN KEY (`instrumentid`) REFERENCES `instrument` (`instrumentid`)
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=latin1;
```

```
-- Dumping structure for table oms.buystocks
CREATE TABLE IF NOT EXISTS `buystocks` (
  `buystock_id` int(100) NOT NULL AUTO_INCREMENT,
  `clientid` varchar(100) NOT NULL,
  `instrumentid` varchar(50) NOT NULL,
  `quantity` int(11) NOT NULL,
  `price` double NOT NULL,
  `buy_date` datetime NOT NULL,
  PRIMARY KEY (`buystock_id`),
  KEY `buy_stocks_ibfk_1` (`clientid`),
  KEY `buy_stocks_ibfk_2` (`instrumentid`),
  CONSTRAINT `buy_stocks_ibfk_1` FOREIGN KEY (`clientid`) REFERENCES `client` (`clientid`),
  CONSTRAINT `buy_stocks_ibfk_2` FOREIGN KEY (`instrumentid`) REFERENCES `instrument` (`instrumentid`)
) ENGINE=InnoDB AUTO_INCREMENT=17 DEFAULT CHARSET=latin1;
```

```
-- Dumping structure for table oms.client
CREATE TABLE IF NOT EXISTS `client` (
  `clientid` varchar(100) NOT NULL,
  `clientname` varchar(100) NOT NULL,
  `transactionlimit` double NOT NULL,
  `custodianid` varchar(100) NOT NULL,
  PRIMARY KEY (`clientid`),
  KEY `client_ibfk_1` (`custodianid`),
  CONSTRAINT `client_ibfk_1` FOREIGN KEY (`custodianid`) REFERENCES `custodian` (`custodianid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
-- Dumping structure for table oms.instrument
CREATE TABLE IF NOT EXISTS `instrument` (
  `instrumentid` varchar(50) NOT NULL,
  `instrumentname` varchar(100) NOT NULL,
  `facevalue` int(10) DEFAULT NULL,
  `expirydate` datetime DEFAULT NULL,
  `quantity` int(11) NOT NULL,
  PRIMARY KEY (`instrumentid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

5.

- **Stored Procedures**: Optimize complex operations like inserting transactions and updating client portfolios.

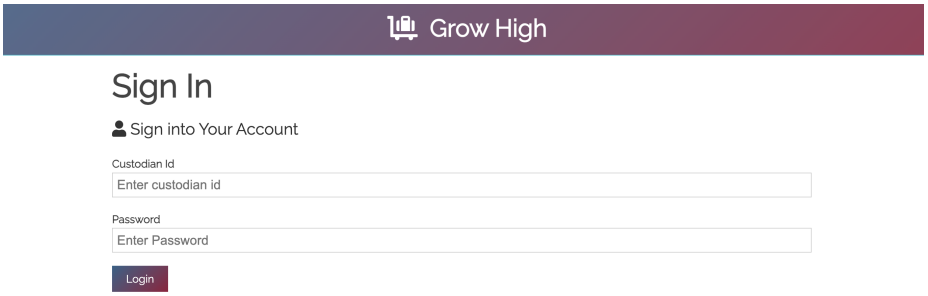
- **GetUniqueTransactionID**: Generates unique IDs for transactions.
- **InsertBuyClient/InsertSellClient**: Handles buy and sell requests while updating relevant tables.
- **UpdateBuyerRecord/UpdateSellerRecord**: Updates client portfolios post-transaction.

- **Indexes**: Speed up query execution for frequently accessed columns like clientid and instrumentid.

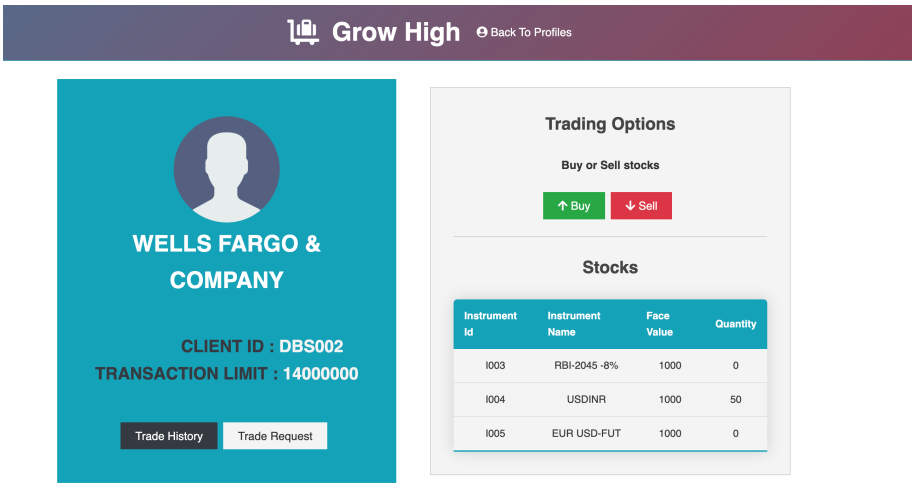
FUNCTIONALITY DETAILS

Basic Functions

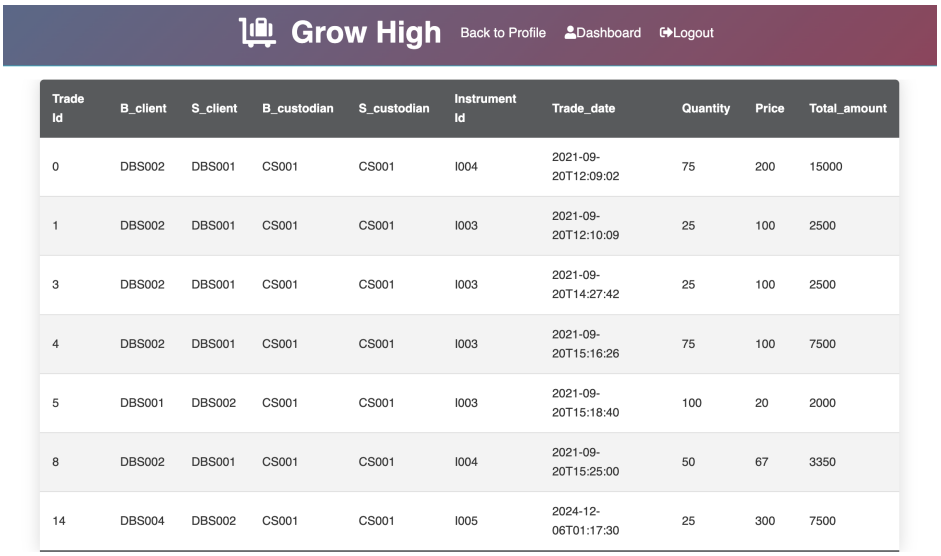
1. **User Authentication**: Secure login with credential validation.



2. **Portfolio Management**: View and manage stocks, including holding details.

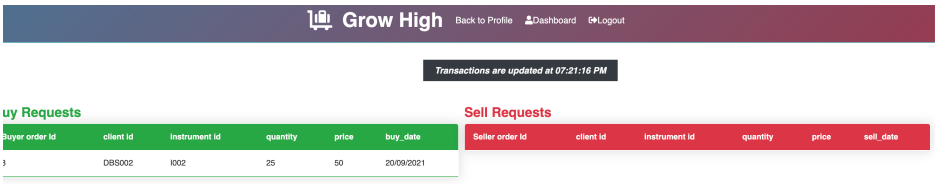


3. **Trade History**: Logs and displays all past transactions.



Trade Id	B_client	S_client	B_custodian	S_custodian	Instrument Id	Trade_date	Quantity	Price	Total_amount
0	DBS002	DBS001	CS001	CS001	I004	2021-09-20T12:09:02	75	200	15000
1	DBS002	DBS001	CS001	CS001	I003	2021-09-20T12:10:09	25	100	2500
3	DBS002	DBS001	CS001	CS001	I003	2021-09-20T14:27:42	25	100	2500
4	DBS002	DBS001	CS001	CS001	I003	2021-09-20T15:16:26	75	100	7500
5	DBS001	DBS002	CS001	CS001	I003	2021-09-20T15:18:40	100	20	2000
8	DBS002	DBS001	CS001	CS001	I004	2021-09-20T15:25:00	50	67	3350
14	DBS004	DBS002	CS001	CS001	I005	2024-12-06T01:17:30	25	300	7500

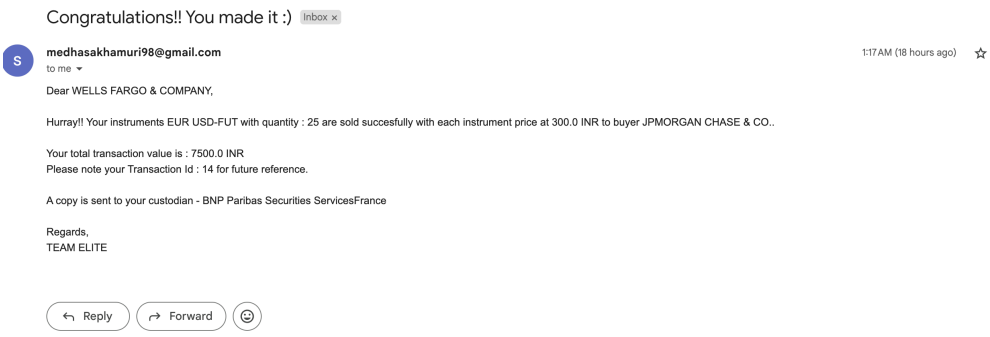
4. **Buy/Sell Stocks**: Place trade requests and execute transactions.



5. **Real-Time Updates**: Dynamic tracking of trade requests.

Advanced Functions

- **Email Notifications**: Sends automated transaction confirmations to clients.



- **Stored Procedures**: Efficiently handle complex trade executions, ensuring seamless integration between database and application logic.

- **Indexing**: Speeds up search queries for stock details and transaction history.

IMPLEMENTATION DETAILS

Languages and Platforms

- **Frontend**: AngularJS with HTML, CSS, and TypeScript for a modern, responsive user interface.
- **Backend**: Spring Boot for RESTful API development and service orchestration.
- **Database**: MariaDB managed through DBeaver for query execution and data organization.

Integration

The frontend communicates with the backend via REST APIs, which, in turn, interact with the database using Spring Data JPA repositories. This architecture ensures efficient data flow and modular development.

Code Repository

The project's codebase is hosted on Dropbox

Experiences

Learnings

- Gained insights into database normalization and the importance of designing efficient schemas.
- Learned to implement stored procedures for complex transaction handling, ensuring data consistency and integrity.
- Developed skills in integrating backend logic with a responsive frontend using REST APIs.

Challenges and Solutions

- **Race Conditions in Transactions**: Resolved by using locks and the FCFS approach in stored procedures.
- **Real-Time Updates**: Implemented dynamic updates through Angular's two-way data binding.

Future Scope

- Integrate predictive analytics for stock recommendations.
- Enhance user experience with real-time market data and advanced visualization tools.
- Improve system scalability by optimizing database queries and API endpoints.

REFERENCES

1. Documentation for AngularJS and Spring Boot frameworks.
2. MariaDB official documentation for database and stored procedure implementation.
3. Resources on relational database design and normalization.