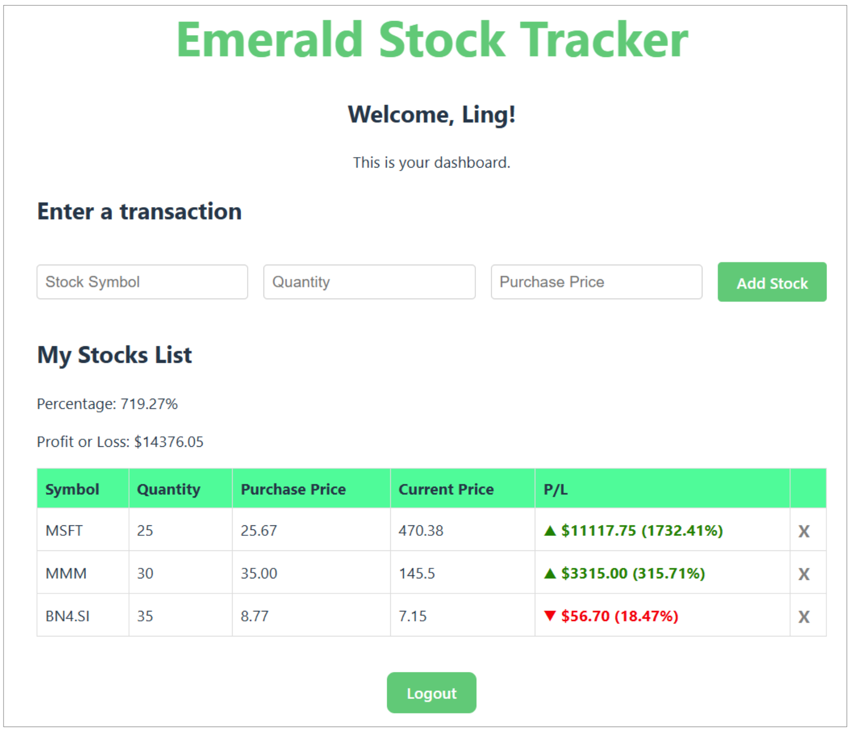
Workshop: Creating a simple stock tracker app with React, NodeJs and MySQL

Date: 9 June 2025



Emerald Stock Tracker

Welcome, Ling!

This is your dashboard.

Enter a transaction

My Stocks List

Percentage: 719.27%

Profit or Loss: $14376.05

Symbol	Quantity	Purchase Price	Current Price	P/L	
MSFT	25	25.67	470.38	▲ $11117.75 (1732.41%)	X
MMM	30	35.00	145.5	▲ $3315.00 (315.71%)	X
BN4.SI	35	8.77	7.15	▼ $56.70 (18.47%)	X

Activities:

1. Download repository from Github
2. Set up environment variables and install libraries
3. Create database and tables
4. Launch backend server and frontend
5. Retrieve stock prices
6. Calculate and display individual Profit or Loss and percentage
7. Calculate overall PnL and percentage

Activity 1:

1. Create a project folder. (can create folder in desktop)
2. Open the folder using VS Code
3. CTRL + Shift + ` to open command line in VS Code
4. Clone starting project repository from Github into project folder using
   git clone https://github.com/nchinling/dft-stock-tracker.git
   (Alternative: Download zipped folder from https://github.com/nchinling/dft-stock-tracker.git. Extract and place in project folder)

Activity 2:

1. Access downloaded repository folder
2. Access stock-tracker-frontend folder from command line. Install library by typing 'npm install'
3. Access stock-tracker-backend folder from command line. Install library by typing 'npm install'
4. Create .env file at root (first-level) of stock-tracker-backend for environment variables. Paste below in .env

```
DB_HOST=localhost
DB_USER=root
DB_PASSWORD=<mysql password>
DB_NAME=stock_tracker
```

Notes:

1. stock-tracker-frontend is created using React with Vite (JavaScript build tool). React is a JavaScript library for building user interfaces (frontend). It uses a virtual DOM and component-based approach.

Virtual DOM – faster and more efficient manipulation of HTML DOM by creating a virtual DOM copy, compares with previous version and updates the real DOM with the changes made.

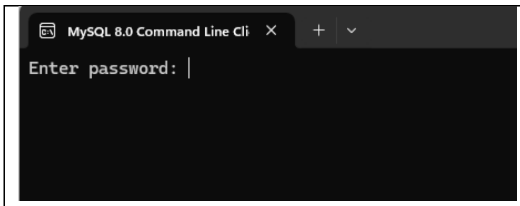
Component-based – reusable, self-contained unit of a UI. React components are used to create and manage DOM elements. Multiple components interact with each other to form a UI.

Vite – a modern frontend build tool for JS applications (i.e. build the React application). Faster development and build time than previous tools.

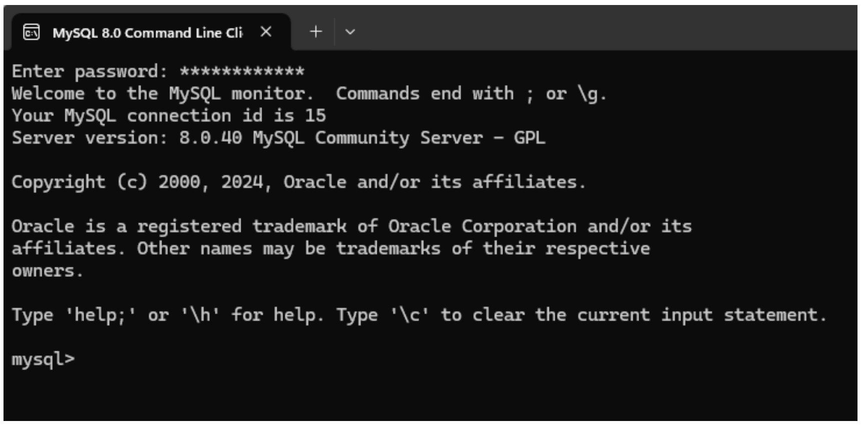
2. List of libraries to be installed are found in package.json.
3. The environment variables here are used to establish database connection.

Activity 3 (database):

1. Connect to MySQL Server using MySQL command-line client. (Alternative: Use MySQL Workbench)



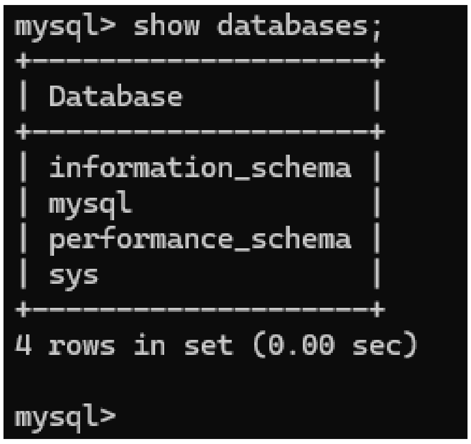
```
Enter password:
```



```
Enter password: ************
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 8.0.40 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

2. Create database

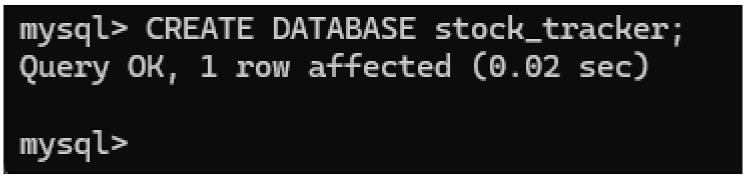
Enter *show databases;* to list existing databases



```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)

mysql>
```

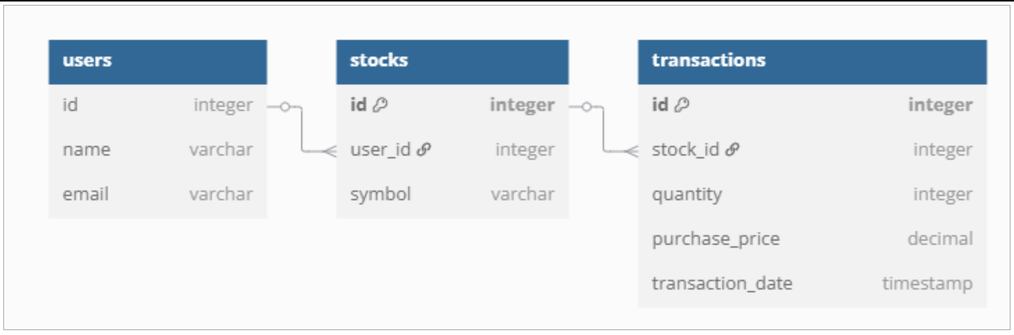
CREATE DATABASE <database_name>; i.e.

CREATE DATABASE stock_tracker;



```
mysql> CREATE DATABASE stock_tracker;
Query OK, 1 row affected (0.02 sec)

mysql>
```

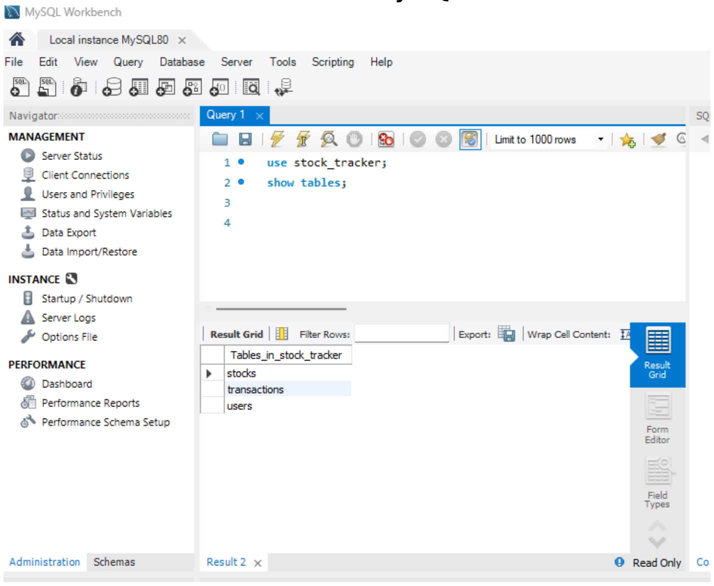
3. Create tables



```
# Access database
USE your_database_name;

i.e. USE stock_tracker;

# Create “users” table
CREATE TABLE users (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   email VARCHAR(255) UNIQUE NOT NULL
);

# Insert values into “users” table
INSERT INTO users (name, email) VALUES
('Ling', 'ling@gmail.com');

# Create “stocks” table
CREATE TABLE stocks (
   id INT AUTO_INCREMENT PRIMARY KEY,
   user_id INT NOT NULL,
   symbol VARCHAR(10) NOT NULL,
   FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);

# Create “transactions” table
CREATE TABLE transactions (
   id INT AUTO_INCREMENT PRIMARY KEY,
   stock_id INT NOT NULL,
   quantity INT NOT NULL,
   purchase_price DECIMAL(10,2) NOT NULL,
   transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (stock_id) REFERENCES stocks(id) ON DELETE CASCADE
);
```

Notes:

1. SQL queries can also be executed in MySQL Workbench.



```
use stock_tracker;
show tables;
```

```
Tables_in_stock_tracker
stocks
transactions
users
```

Activity 4 (Run application):

1. Frontend: npm run dev
2. Backend: npx nodemon server.js
3. Launch app on browser: http://localhost:5173/

Notes:

1. ‘npm run dev’ is the Vite command to start server
2. ‘npx nodemon’ command enables automatic restart when code changes are saved.

Activity 5 (Retrieve prices):

stock-tracker-backend

1. Modify fetch stocks in stockRoutes.js
   a) Uncomment **code snippet number 2** in activity5_getprice.js
   b) Paste the import statement at the top of stockRoutes.js to import the new function (created in step 3).
   c) Paste the try-catch block below the commented ‘Participants’ line in stockRoutes.js. Remove the existing placeholder code.
2. In stock-tracker-backend, create a ‘services’ folder.
3. Create addStockPrices.js file in ‘services’ folder.
   a) Uncomment **code snippet number 1** in activity5_getprice.js

b) Paste the uncommented code into addStockPrices.js

(refer to activity5_getprice.js in activities folder in stock-tracker-backend for codes)
Yahoo-finance2 documentation: https://www.npmjs.com/package/yahoo-finance2

Activity 6 (Calculate individual P&L & %P&L):

stock-tracker-backend

1. Modify fetch stocks in stockRoutes.js
   a) Uncomment **code snippet number 2** in activity6_calcpnl.js
   b) Paste the import statement at the top of stockRoutes.js to import the new function (created in step 3).
   c) Replace the existing try-catch block (created in Activity 5) with the one in activity6_calcpnl.js. The new try-catch block has a new function to calculate individual P&L with its percentage and has a new variable 'stockListWithPnLandPrices' which is returned in the response.

2. Create calculatePnL.js file in 'services' folder.
   a) Uncomment **code snipper number 1** in activity6_calcpnl.js
   b) Paste the uncommented code into calculatePnL.js

stock-tracker-frontend

1. Display data in frontend by modifying StockList.jsx
   a) Uncomment the code snippet in PnLDisplay.jsx
   b) In StockList.jsx, replace <td>Placeholder</td> with the uncommented code.

(refer to activity6_calcpnl.js in activities folder in stock-tracker-backend and PnLDisplay.jsx in activities folder in stock-tracker-frontend)

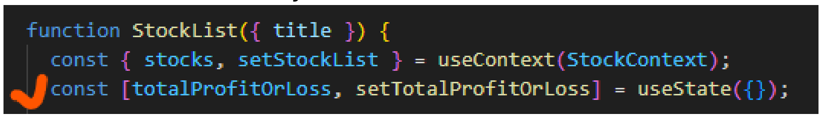
Activity 7 (Calculate overall profit and %):

stock-tracker-frontend

1. Create TotalProfitOrLoss.jsx file in components folder
   a) Uncomment TotalProfitOrLoss function in TotalPnLDisplay.jsx.
   b) Paste uncommented code into TotalProfitOrLoss.jsx

2. Create useEffect and useState hooks in StockList.jsx
   a) Uncomment the import statement in TotalPnLDisplay.jsx.
   b) Paste the import statement at the top of StockList.jsx to import the new function (created in step 1).
   c) Uncomment the totalProfitOrLoss state variables (snippet with UseState()) in TotalPnLDisplay.jsx.

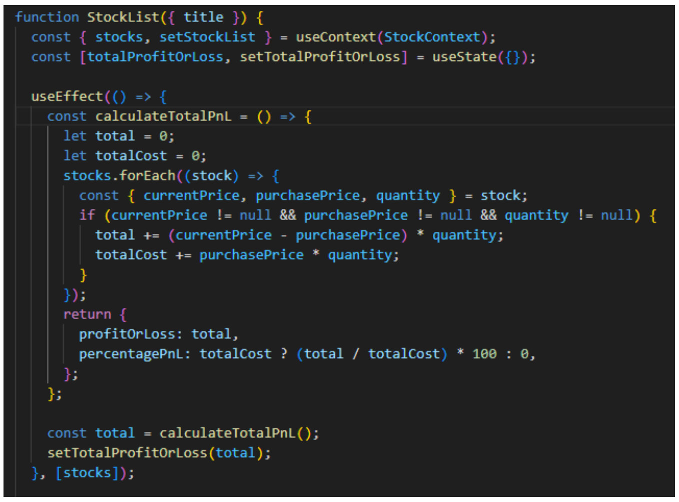
d) Paste the totalProfitOrLoss state variable snippet just below the StockList function in StockList.jsx



```
function StockList({ title }) {
  const { stocks, setStockList } = useContext(StockContext);
  const [totalProfitOrLoss, setTotalProfitOrLoss] = useState({});
```

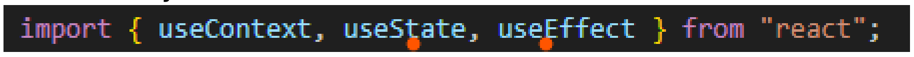
e) Uncomment the useEffect code snippet in TotalPnLDisplay.jsx.

f) Paste the useEffect code snippet below the totalProfitOrLoss state variable.



```
function StockList({ title }) {
  const { stocks, setStockList } = useContext(StockContext);
  const [totalProfitOrLoss, setTotalProfitOrLoss] = useState({});

  useEffect(() => {
    const calculateTotalPnL = () => {
      let total = 0;
      let totalCost = 0;
      stocks.forEach((stock) => {
        const { currentPrice, purchasePrice, quantity } = stock;
        if (currentPrice != null && purchasePrice != null && quantity != null) {
          total += (currentPrice - purchasePrice) * quantity;
          totalCost += purchasePrice * quantity;
        }
      });
      return {
        profitOrLoss: total,
        percentagePnL: totalCost ? (total / totalCost) * 100 : 0,
      };
    };

    const total = calculateTotalPnL();
    setTotalProfitOrLoss(total);
  }, [stocks]);
```

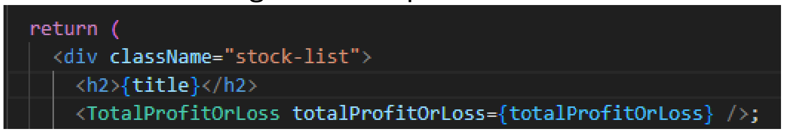
g) To use the newly introduced hooks, import useState and useEffect from the React library.



```
import { useContext, useState, useEffect } from "react";
```

3. Insert TotalProfitOrLoss component in StockList.jsx

   a) Paste <TotalProfitOrLoss totalProfitOrLoss={totalProfitOrLoss} />; below the title with h2 tag. The total profit or loss will be shown below the title.



```
return (
  <div className="stock-list">
    <h2>{title}</h2>
    <TotalProfitOrLoss totalProfitOrLoss={totalProfitOrLoss} />;
```

(refer to TotalPnLDisplay.jsx in activities folder in stock-tracker-frontend)

Notes:

1. States refer to data that can change over time (mutable). It affects a component's behaviour and rendering.
2. useState, useContext and useEffect are React hooks that manage states and side effects in components.
3. useState – declares and updates a state in the local component.
4. useEffect – runs on the side in a component when a state changes.
5. useContext – manages global state across components. Components can access shared values.

End