FOX OF HOOD

PROJECT STATUS REPORT | MIDSEMESTER

Team B

Manila Aryal, Grace Frizzell, Tamsyn Evezard, Ngoc Bui, Berkan Guven

10/29/2024 6:00pm

https://github.com/ettany/FOH_PSE.git

Roles

- **Grace:** Database & Transaction Lead
- **Manila:** Activity Log & Password Hashing Lead
- **Ngoc:** Visualization & CAPTCHA Lead
- **Tamsyn:** Authentication & API Lead
- **Berkan:** Facial Recognition Lead

Project Info

How our team operates:

- Zoom meetings (planning, code reviews, milestone reviews).
- WhatsApp & Email (communicating pushed code & updates to github)
- In-Person Meetings (informal progress reports before/after classes)

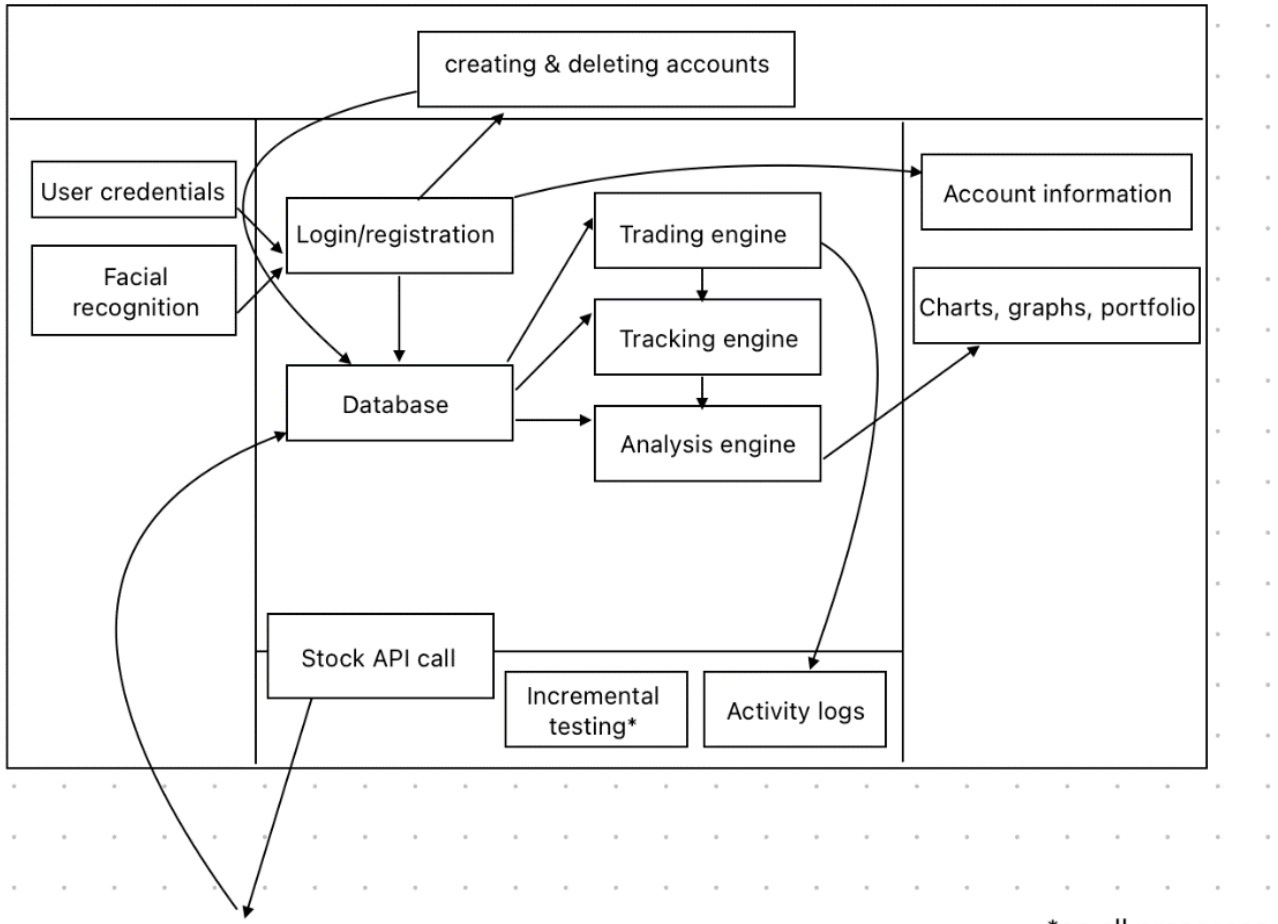
Development process used:

- Agile/incremental development

Tools used:

- **Planning:** Asana to assign tasks and measure progress
- **Development:** VS Code, FIGMA (UI visualization), shared GitHub repository

Software Architecture



FOX OF HOOD – Architecture Overview

- **User Interface Layer**
  - **Frontend**: This is what users see and interact with, like the buttons, charts, and stock information. It's designed to be easy to use.
  - **Login and Security**: Users log in with a username and password (CAPTCHA for extra security). Face login is a feature.
- **Core Application Parts**
  - **User Management**: Controls user accounts (like creating, editing, and deleting accounts). Admins (special users) have extra control over accounts.
  - **Portfolio Management**: This is the main tool for users to manage their stocks. It lets users add, update, and sell stocks in their portfolios.
  - **Charts and Graphs**: Creates charts that show stock performance, money trends, and more.
  - **Activity Logging**: Records every action users take, like logging in, adding stocks, or making changes, so admins can track usage.
- **Data Management Layer**
  - **Database**: A place to save user data, portfolios, logs, and other important information permanently.
- **External Integrations**
  - **Stock Price API**: Connects to online service to get the latest stock prices so users see real-time updates.
  - **CAPTCHA Service**: Adds extra login security to prevent bots from logging in.
- **Backend Structure**
  - **Backend API**: This connects the front-end (user's screen) with the back-end (where the app's work happens), so all parts of the app can share data.

Software Architecture | Traceability

- **Example:** "multiuser application provided by a traditional mechanism (e.g., username and password)"
  - Authentication | REQ-3: Implement logic for identical username prevention

```python
def register():
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]

        # Check if the username already exists
        db_conn = db.get_db()
        existing_user = db_conn.execute(
            "SELECT * FROM user WHERE username = ?",
            (username,),
        ).fetchone()

        if existing_user:
            flash("Username already taken. Please choose a different one.")
            return redirect(url_for("register"))

        # Insert user into the database
        db_conn.execute(
            "INSERT INTO user (username, password, totalCash) VALUES (?, ?, ?)",
            (username, password, 100000),
        )
        db_conn.commit()

        return redirect(url_for("login"))

    return render_template("register.html")
```

```python
def register():
    if request.method == "POST":
        data = request.get_json()  # Get JSON data from the request
        username = data.get('username')
        password = data.get('password')

        if not username or not password:
            return jsonify({"error": "Username and password are required."}), 400
        # Check if the username already exists
        db_conn = get_db()
        existing_user = db_conn.execute(
            "SELECT * FROM user WHERE username = ?",
            (username,)
        ).fetchone()
        if existing_user:
            return jsonify({"error": "Username already taken"}), 400

        # Hash the password
        hashed_password = generate_password_hash(password, method='pbkdf2:sha256')

        try:
            # Insert user into the database
            db_conn.execute(
                "INSERT INTO user (username, password, totalCash) VALUES (?, ?, ?)",
                (username, hashed_password, 100000)
            )
            db_conn.commit()
            return jsonify({"message": "User registered successfully!"}), 201
        except Exception:
            return jsonify({"error": "Registration failed. Please try again."}), 500
```

Developed Software | Selling stocks

```python
# API route to sell stocks
@transaction_bp.route('/sell', methods=['POST'])
def sell():
    data = request.get_json()
    ticker = data.get("ticker")
    numShares = int(data.get("numShares"))
    session['username']=data.get("username")
    stock = yf.Ticker(ticker)
    stock_info = stock.info

    totalProfit = numShares * stock_info.get("currentPrice", 0)

    db_conn = get_db()
    session['username']='ngoc'
    id = db_conn.execute(
        "SELECT id FROM user WHERE username = ?", (session['username'],)).fetchone()
    if id is None:
        return jsonify({"error": "User not found"}), 404

    currentShares = db_conn.execute(
        "SELECT numShares FROM portfolio WHERE id = ? AND ticker = ?", (id['id'], ticker)).fetchone()

    if currentShares is None:
        return jsonify({"error": "No shares found in portfolio"}), 404
```

```python
if currentShares['numShares'] >= numShares:
    # Update user's total cash
    db_conn.execute(
        "UPDATE user SET totalCash = totalCash + ? WHERE id = ?", (totalProfit, id['id']))
    # Log the selling action
    db_conn.execute(
        "INSERT INTO eventLog (id, eventName, stockBought) VALUES (?, ?, ?)", (id['id'], 'Sold', ticker))
    # Update shares in portfolio
    db_conn.execute(
        "UPDATE portfolio SET numShares = numShares - ? WHERE id = ? AND ticker = ?", (numShares, id['id'], ticke
    db_conn.commit()
    return jsonify({"message": "Stock sold successfully", "totalProfit": totalProfit}), 200
else:
    return jsonify({"error": "Not enough shares to sell"}), 400
```

- Access stocks via user input of ticker and number of shares
- Check to see if minimum number of shares are owned
- Record total profit
- Add total profit to cash total
- Remove shares from portfolio
- Add action to the event log

Developed Software | Database

```sql
DROP TABLE IF EXISTS user;
DROP TABLE IF EXISTS portfolio;
DROP TABLE IF EXISTS eventLog;

CREATE TABLE user (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  username VARCHAR(100) UNIQUE NOT NULL,
  password VARCHAR(100) NOT NULL,
  totalCash DECIMAL(15,2)
);

CREATE TABLE portfolio (
    ticker VARCHAR(10) UNIQUE NOT NULL,
    numShares INTEGER,
    id INTEGER,
    FOREIGN KEY (id) REFERENCES user(id)
);

CREATE TABLE eventLog (
    id INTEGER,
    eventName TEXT CHECK(eventName IN ('Bought', 'Sold', 'Logged on', 'Logged out')) NOT NULL,
    stockSold VARCHAR(10),
    stockBought VARCHAR(10),
    date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (id) REFERENCES user(id)
);
```

- User table
  - Contains user ID, username, password, and tracks total cash in account
- Portfolio
  - Stores stocks and number of shares owned by the user
- Event log
  - Tracks buying, selling, and log events.

Developed Software

```python
def login():
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]

        # Check the credentials against the database
        try:
            db_conn = db.get_db()
            user = db_conn.execute(
                "SELECT * FROM user WHERE username = ? AND password = ?",
                (username, password),
            ).fetchone()

            if user:
                session["username"] = username
                if username == "administration":
                    return redirect(url_for("admin"))
                else:
                    session["totalCash"] = user["totalCash"]
                    return redirect(url_for("index"))
            else:
                flash("Invalid Credentials.")
                return redirect(url_for("login"))

        except Exception as e:
            print(f"Error: {e}")
            return "Database connection error"

    return render_template("login.html")
```

Authentication (username/password)

- Role-based access control (user/admin user) to allow/restrict certain users from certain functions
  - Registration: if username already taken, show flash message
  - If username not in the database or invalid credentials, show flash message

Note: this login function has since been adjusted to fix the error we were having with the flask session temporarily (explained and shown in demo).

Accomplished Work

- Username and password authentication for users and admin that maintains session (functionality included in the **register, login,** and **logout** pages)
- Password hashing for account security
- Working database with user, portfolio, and eventLog tables
- **Portfolio** page
  - displaying the specific user and their initial balance of $100,000
  - other hardcoded UI elements for functional visualization
- **Trade** page
  - displaying updated stock prices for 5 stocks with a table and graph
  - ability to search for a specific stock and display its details
  - ability to buy or sell an amount of specific stock
  - graph of searched stock's performance over the last 30 days
- **Log** page
  - Hardcoded UI elements for initial visualization

Accomplished Work

How good is it?

- Progress measurement: we have achieved all milestones that we defined to be ready and working by this Project Status Report
- Well established schedule: although we had a rocky start with progressing with the project and its components, we now have a structured plan and designated leaders within the team which has allowed things to run smoothly recently

Project size:

- Backend: about 350 lines of code
- Frontend:
  - Static: about 210 lines of code
  - Templates: about 367 lines of code

Estimated hours of effort: 60 hours

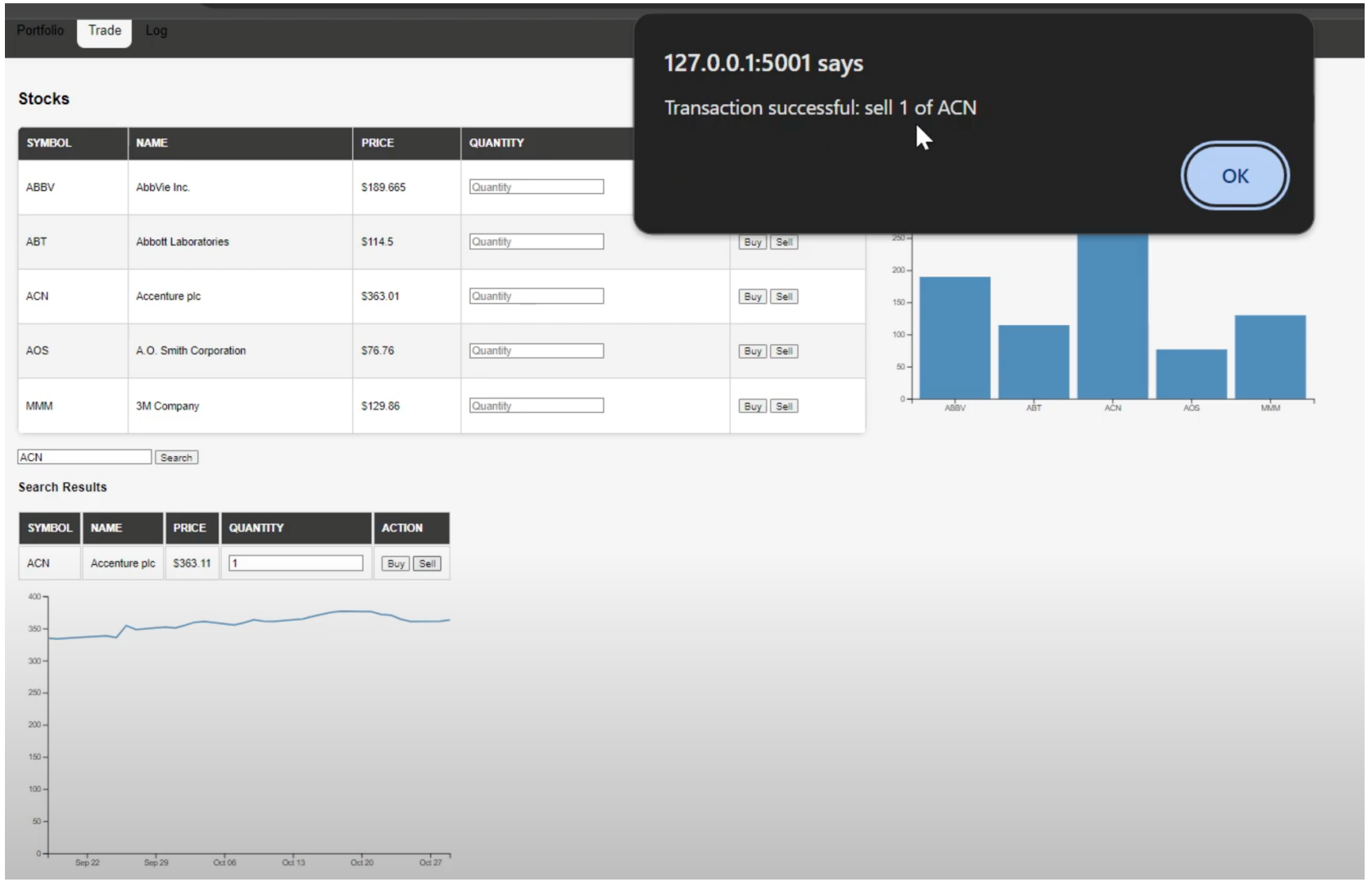
Developing the product to ensure compatibility:

- Python & Flask (backend)
- SQL (database schema)
- Frontend (HTML, CSS, JavaScript)
- API: yahoo finance

Our Product

3 Key Features:

- ✓ Buy & Sell,
- ✓ Stocks (API),
- ✓ Visualization (graphs)



Lessons Learned

- What went right?
  - Using established leadership roles within the project (as shown on slide 2)
  - Roles allowed us to break the project into manageable chunks and gave team members ownership over their specific roles
- What went wrong?
  - Coordination and collaboration: our initial research and planning stage proved inefficient due to the absence of clearly defined roles
  - Difficulties implementing the buy and sell functionality & flask session
- What would you do differently?
  - Establish roles and rules earlier in the SDLC
  - Stick to the languages and frameworks we are most familiar with

Demo

https://www.youtube.com/watch?v=jCZ426may-Q