## Objective

The main goal of this project is to create a practice stock trading web app where people can buy and sell shares with fake money. It's like a stock market game that feels real, but without risking real cash.

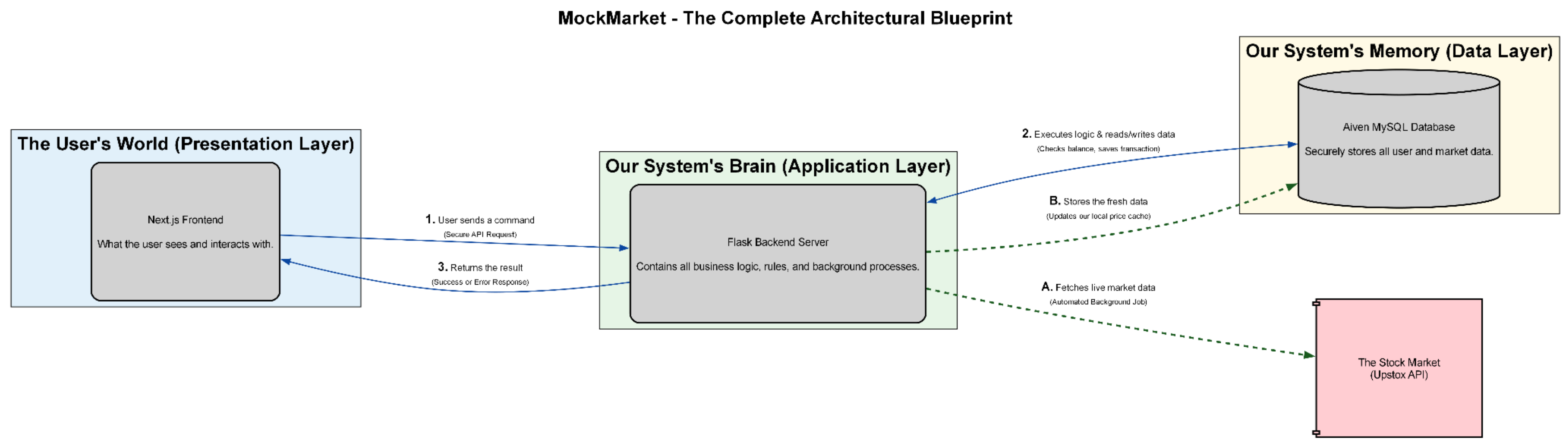
## Purpose

- To help beginners learn how stock trading works in a safe environment.
- To allow users to practice buying/selling stocks, track their profit/loss, and understand portfolio management.
- To build a database driven system that keeps all the records (users, stocks, transactions, holdings) in an organized way.

## Tech Stack

- **Database:** MySQL (hosted on **Aiven** free cloud SQL).
- **Backend:** Flask (Python) connects DB and APIs, handles trading logic.
- **Data Fetcher:** Python script using **Upstox API** for real-time stock prices.
- **Frontend:** Next.js

## System Architecture Overview


MockMarket - The Complete Architectural Blueprint
The User's World (Presentation Layer)
Next.js Frontend
What the user sees and interacts with.
1. User sends a command
(Secure API Request)
3. Returns the result
(Success or Error Response)
Our System's Brain (Application Layer)
Flask Backend Server
Contains all business logic, rules, and background processes.
2. Executes logic & reads/writes data
(Checks balance, saves transaction)
B. Stores the fresh data
(Updates our local price cache)
A. Fetches live market data
(Automated Background Job)
Our System's Memory (Data Layer)
Aiven MySQL Database
Securely stores all user and market data.
The Stock Market
(Upstox API)

### 1. Presentation Layer (Frontend)

- **Tech:** Next.js
- **Role:** User interface for interaction and visualization.
- **Focus:** Displays data from the backend and sends user actions as API requests.

## 2. Application Layer (Backend)

- **Tech:** Flask (Python)
- **Role:** Core logic and API controller.
- **Focus:** Handles authentication, trade logic, and links frontend with database.

## 3. Data Layer (Database)

- **Tech:** MySQL (Aiven)
- **Role:** Secure data storage and retrieval.
- **Focus:** Stores user info, transactions, and portfolio data.

## 4. External Service

- **Tech:** Upstox REST API
- **Role:** Real-time market data provider.
- **Focus:** Supplies live stock prices and market information to the backend.

## Key Data Flows

### Flow 1: The User Action Cycle (Blue Arrows)

**Type:** Synchronous, real-time interaction
**Purpose:** Handles direct user commands like buying or selling a stock.

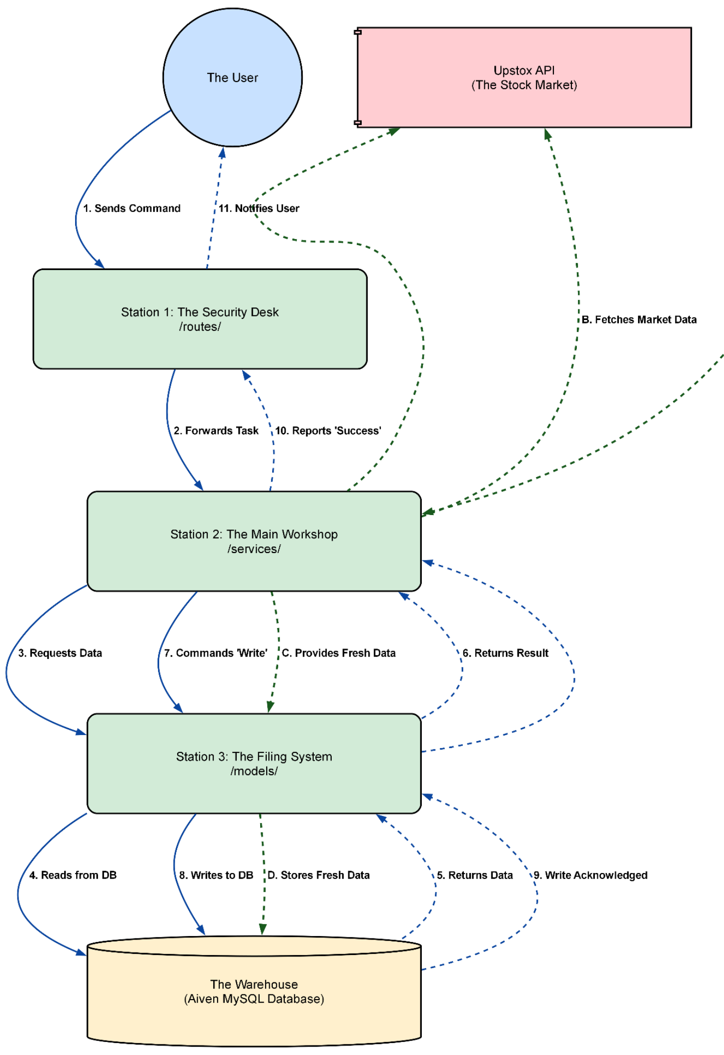
- **Request:** The user initiates an action on the Frontend, which sends a secure API request to the Backend.
- **Process:** The Backend executes business logic (e.g., balance check, cost calculation) and interacts with the Database to update relevant records.
- **Response:** Once complete, the Backend sends the result back to the Frontend, which updates the interface (e.g., "Purchase Successful").

### Flow 2: The Market Data Sync (Green Arrows)

**Type:** Asynchronous, background process
**Purpose:** Keeps market data fresh and accurate.

- **Fetch (A):** On a set schedule, the Backend requests live market data from the Upstox API.
- **Store (B):** The Backend processes the received data and updates the Database, maintaining an up-to-date local cache for the system.

## MockMarket Backend Architecture: The Definitive Blueprint



MockMarket Backend: The Definitive Blueprint

## 1. Core Architecture Overview

The backend is structured into **three internal layers** and **two external actors**, all working together to deliver a scalable and maintainable system.

### A. The Core “Factory” (Flask Backend Application)

**Station 1: The Security Desk (`/routes/`)**
**Role:** Entry point for all client requests.

**Responsibilities:**

- Handles all incoming API requests from the frontend.
- Validates authentication tokens and input formats.
- Forwards verified requests to the service layer.

*Contains no business logic — purely a security and routing layer.*

**Station 2: The Main Workshop (`/services/`)**

**Role:** Core business logic engine.

**Responsibilities:**

- Executes actions like user registration, stock purchases, and balance checks.
- Coordinates database operations and external API calls.
- Ensures logical correctness and consistency across the system.

*Acts as the brain of the application.*

**Station 3: The Filing System (`/models/`)**

**Role:** Database interface and ORM layer.

**Responsibilities:**

- Manages all database interactions through clean, Python-based models.
- Provides functions for querying, inserting, and updating records.

*The only layer allowed to directly communicate with the database.*

## B. The Warehouse (Data Layer)

**Technology:** Aiven MySQL Database
**Role:** Persistent storage for all application data.
**Responsibilities:**

- Stores user details, transaction logs, and cached market data.
- Ensures data security, reliability, and high performance.

*Serves as the system’s permanent memory.*

## C. External Actors & Services

### 1. The User

- The end-user interacting through the web frontend (Next.js).
- Initiates commands that trigger synchronous backend flows.

### 2. The Scheduler

- A background worker running on a timed schedule.
- Triggers automated processes like market data syncs.

### 3. The Upstox API

- External data source for real-world market prices and stock metadata.
- Provides live updates to maintain the accuracy of local data.

## 2. Key Data Flows

The diagram defines **two major operational flows** that run the MockMarket system.

### Flow 1: User Action Cycle (Blue Arrows)

**Type:** Synchronous, real-time
**Purpose:** Handles direct user commands like buying or selling stocks.

1. **User Action:** The frontend sends a secure API request to the backend's `/routes/`.
2. **Validation:** The request is authenticated and passed to the `/services/` layer.
3. **Processing:** Business logic executes; data is read or written via `/models/`.
4. **Database Access:** The database stores or retrieves the requested information.
5. **Response:** The backend returns a structured response to the frontend (e.g., "Purchase Successful").

*This flow represents user-driven operations that require instant feedback.*

### Flow 2: Automated Data Sync (Green Arrows)

**Type:** Asynchronous, background process
**Purpose:** Keeps market data current and consistent.

A. **Trigger:** The Scheduler initiates a background task at fixed intervals.
B. **Fetch:** The `/services/` layer requests live market data from the Upstox API.
C. **Process:** Received data is validated and formatted.
D. **Store:** Updated market data is saved in the database through `/models/`.

*This flow ensures local data remains synchronized with real-world markets, even without user activity.*

```
/MockMarket/
├── backend/
│   ├── .venv/            # The Python virtual environment (ignored by Git)
│   ├── routes/           # Station 1: The "Security & Sorting Desk"
│   │   ├── __init__.py      # Makes the 'routes' folder a Python package
│   │   ├── user_routes.py  # Handles /api/register, /api/login, etc.
│   │   └── trade_routes.py  # Handles /api/trade/buy, /api/trade/sell, etc.
│   │
│   ├── services/          # Station 2: The "Main Workshop"
│   │   ├── __init__.py      # Makes the 'services' folder a Python package
│   │   ├── user_service.py  # Logic for creating users, checking passwords
│   │   └── trade_service.py # Logic for executing a buy or sell order
│   │
│   ├── models/            # Station 3: The "Filing System"
│   │   ├── __init__.py      # Makes the 'models' folder a Python package
│   │   ├── user_model.py    # Defines the User table
│   │   ├── stock_model.py   # Defines Stock, StockPrice, StockHistory tables
│   │   └── trade_model.py   # Defines Transaction, Portfolio tables
│   │
│   ├── utils/             # The "Toolbox"
│   │   ├── __init__.py      # Makes the 'utils' folder a Python package
│   │   └── token_utils.py   # Helper functions for creating/validating JWTs
│   │
│   ├── app.py             # The "Assembler" - Initializes and starts everything
│   ├── scheduler.py        # The "Automated Worker" - Runs background jobs
│   ├── config.py          # The "Blueprints" - All configuration settings
│   ├── .env               # The "Safe" - All secret keys (API, DB password)
│   ├── requirements.txt     # List of all Python packages needed for the project
│   └── .gitignore          # Tells Git which files to ignore
│
└── frontend/             # The Next.js frontend project will live here
```