AWS-Powered Stock Portfolio Management Application

A Full-Stack Cloud-Native Implementation

Team Members:

Rudraksh Chaurey I Rishabh Kumar I Kritika Chhabra I Ujjwal Jain I Dev Pratap Singh I Adarsh Arora

Our Vision: A Secure, Scalable, Real-Time Platform for Modern Investors

We set out to engineer a full-stack portfolio management system that combines the agility of cloud-native services with the robustness of enterprise-grade security and data management. Our solution is built on three foundational principles:

Real-Time Data

Empowering users with up-to-the-minute market information for timely, informed decisions.

Robust Security

Implementing a multi-layered, token-based security model to protect user data and ensure trust.

Scalable Infrastructure

Building on a cloud foundation that can grow effortlessly with user demand without compromising performance.

Core Project Modules

Our application is built on a modular, services-oriented architecture with a clear separation of concerns.

User-Facing Frontend

The interactive and responsive user interface built with **React**. This is the entry point for all user interactions, from registration to portfolio management.

Core Services Backend

The brain of the application. A **Spring Boot** service that houses all business logic, manages data, and exposes a secure REST API.

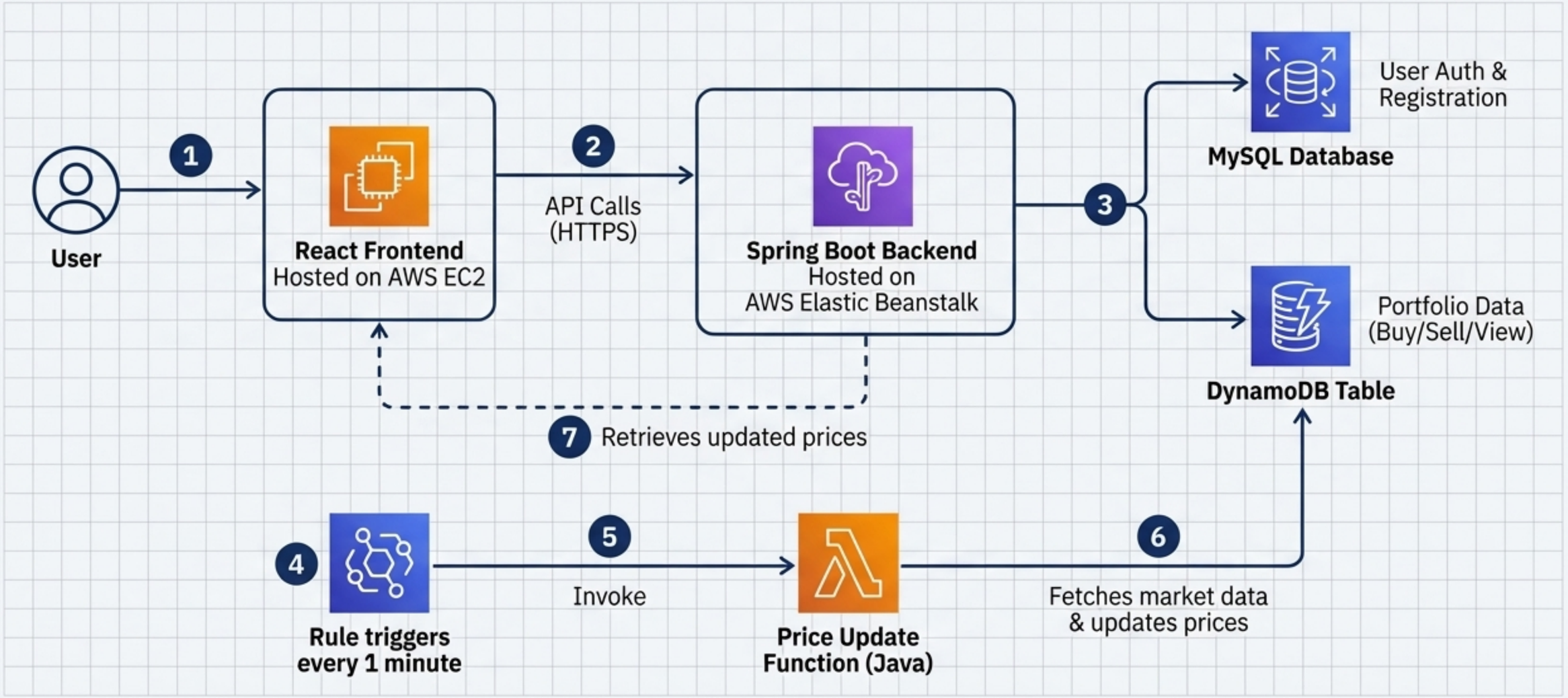
Data & Persistence Layer

A hybrid data storage solution using **AWS RDS** for structured user data and **AWS DynamoDB** for dynamic, high-volume portfolio data.

Cloud Infrastructure & Automation

The foundation of our system. We leverage **AWS** services like **Elastic Beanstalk**, **EC2**, **Lambda**, and **EventBridge** for deployment, hosting, and automation.

A Decoupled, Cloud-Native Architecture

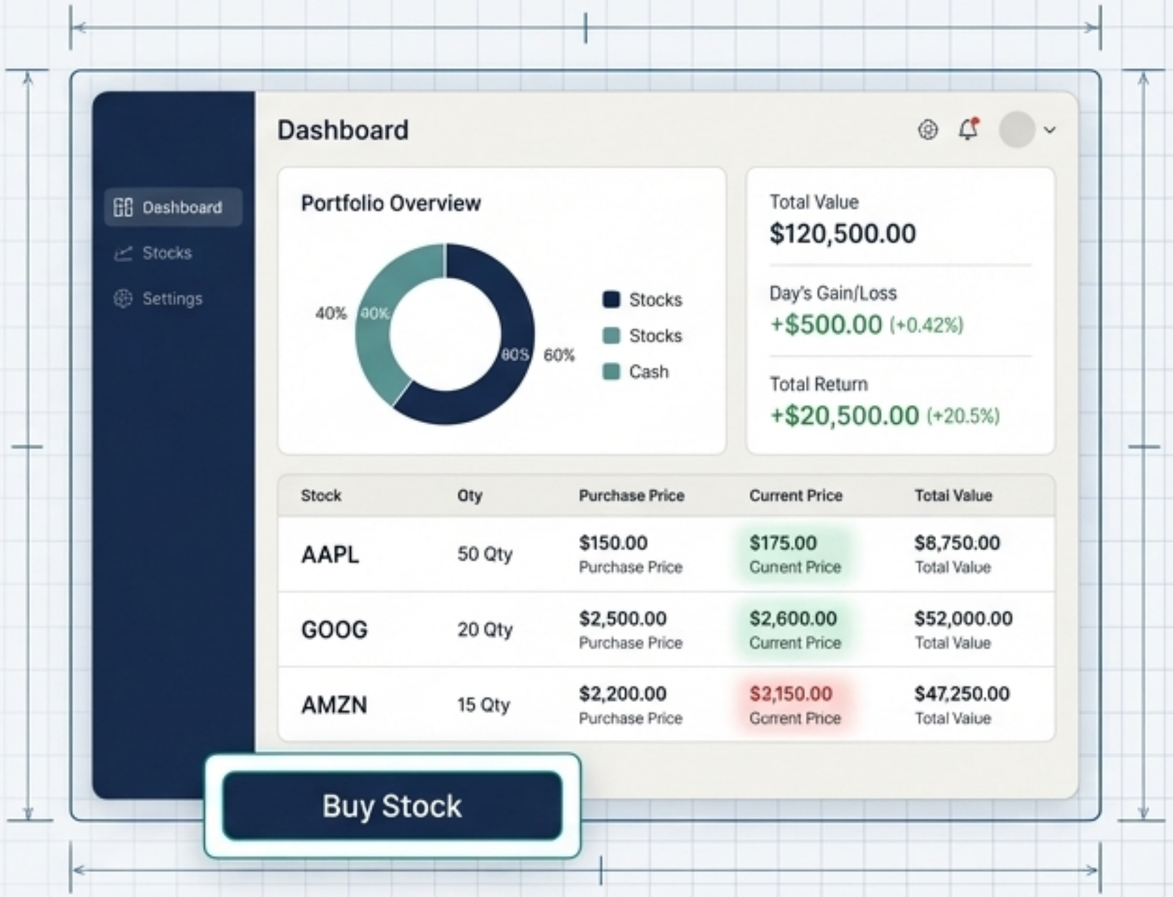


The Frontend Experience: Built with React

The frontend is a single-page application (SPA) built with React, ensuring a fluid and modern user experience.

Key User Journeys:

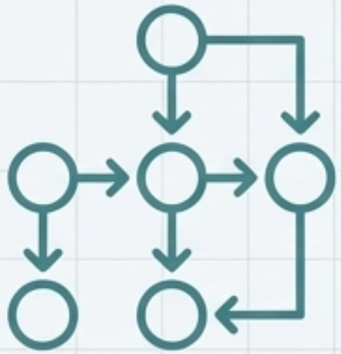
- **Seamless Onboarding**: A streamlined registration process using controlled forms to collect user details like name, email, DOB, risk appetite, and investment goals.
- **Secure Authentication**: A dedicated login flow that securely transmits credentials to the backend for verification.
- **Interactive Dashboard**: The central hub where users view their portfolio, see real-time stock prices, and receive suggestions to buy stocks if their portfolio is empty.
- **Effortless Trading**: Intuitive modals for buying and selling stocks, with all transactions immediately reflected in the user's portfolio via API calls.



The Backend Engine: A Robust Spring Boot Core

The backend is a powerful Spring Boot application responsible for encapsulating all business logic and providing a secure, stateless RESTful API.

Core Responsibilities:



Business Logic:

- Manrope for user registration, authentication, stock transactions (buy/sell), and portfolio calculations.

Data Orchestration:

- Acts as the single intermediary between the client and our hybrid database layer, ensuring data consistency across RDS and DynamoDB.

Security Enforcement:

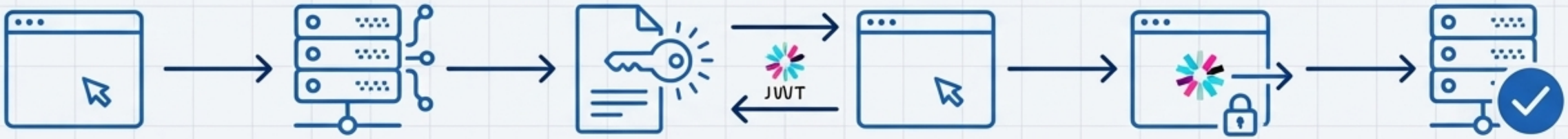
- Implements authentication and authorization logic, protecting user data and system resources through a role-based access model.

A Well-Defined, Role-Based API Contract

Global APIs (Public Access)	User APIs (User Role Required)	Admin APIs (Admin Role Required)
POST /register	GET /user-portfolio/{userId}	POST /admin/stocks/add
POST /login	POST /portfolio/update	DELETE /admin/deleteStock/{stockId}
GET /stocks		PUT /admin/stocks/update
		GET /admin/allUsers
		DELETE /admin/deleteUser/{email}
		PUT /admin/user/update

Securing the Application with JWT and Role-Based Access Control

We implemented a modern, token-based security model to ensure our application is both secure and scalable.



1. Login Request

The user submits their email and password from the React frontend.

2. Credential Validation

Spring Boot validates the credentials against user records in the AWS RDS database.

3. Token Generation

Upon success, Spring Security generates a JSON Web Token (JWT) containing the user's email and role (e.g., 'USER' or 'ADMIN').

4. Token Response

The JWT is sent back to the React client.

5. Authorized Requests

For all subsequent API calls to protected endpoints, the React client includes this JWT in the `Authorization` header.

6. Backend Verification

Spring Boot validates the JWT on every request, granting access only if the token is valid and the user's role permits the action.

The Right Tool for the Right Job: Our Hybrid Database Strategy

A single database model could not meet our diverse needs for data integrity, performance, and scalability. We adopted a hybrid strategy to leverage the best of both worlds.

AWS RDS (MySQL) for Relational User Data

What it stores:
Highly structured user data: userid, firstname, lastname, email, password, dob, risk appetite, etc.

Why we chose it:

- **Data Integrity:** Enforces schema and ACID compliance, crucial for critical user account information.
- **Relational Power:** Ideal for maintaining clear relationships between user attributes.

AWS DynamoDB for Dynamic Portfolio Data

What it stores:
High-volume, frequently updated portfolio data: userid, stockid, purchase_price, quantity.

Why we chose it:

- **Scalability & Performance:** Delivers consistent, single-digit millisecond latency needed for real-time portfolio value display and trading.
- **Flexible Schema:** Easily handles dynamic stock data without complex migrations, supporting fast-paced updates.

Delivering Fresh Data, Every Minute

To provide users with timely market information, we built a serverless, event-driven pipeline that updates stock prices automatically. This design choice offloads the recurring task from our main application server, ensuring the backend remains responsive to user requests.

The One-Minute Update Cycle

1. Trigger

An Amazon EventBridge rule fires a scheduled event every one minute.

2. Invoke

The event triggers our AWS Lambda function, which contains a pre-packaged Java .jar file with the update logic.

3. Process

The Lambda function executes, fetching the latest market prices for all tracked stocks.

4. Update

The function writes the new prices directly into our AWS DynamoDB stocks table.

5. Display

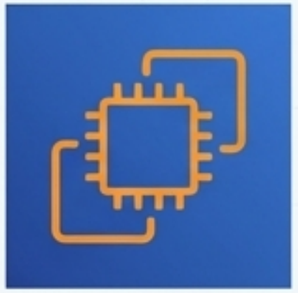
The React frontend, through the Spring Boot API, reads this constantly updated data from DynamoDB, ensuring portfolio values are always current.

From Code to Cloud: A Robust Deployment Strategy

Our infrastructure is designed for reliability, scalability, and ease of management using dedicated AWS services.

Backend Deployment: AWS Elastic Beanstalk

- Our Spring Boot application is packaged as a `.jar` file and deployed to Elastic Beanstalk.
- **Why?** Elastic Beanstalk automates infrastructure provisioning, load balancing, auto-scaling, and health monitoring. This allows us to focus on application code, not server management, and ensures the backend can handle traffic spikes automatically.



Frontend Hosting: AWS EC2

- The production build of our React application is hosted on a configured AWS EC2 instance.
- **Why?** EC2 provides reliable and customizable compute capacity. It is configured to serve the static React files and communicates seamlessly with the Elastic Beanstalk environment.

The Admin Portal: System Management & Control

Beyond the primary user application, we have built a secure administrative portal for complete platform management. Access is restricted to users with the ‘ADMIN’ role, enforced by our JWT security model.

Admin Capabilities:

- **User Management:**
  - View a list of all registered users (`GET /admin/allUsers`)
  - Update user details (`PUT /admin/user/update`)
  - Delete users from the system (`DELETE /admin/deleteUser/{email}`)
- **Stock Market Management:**
  - Add new stocks to be tracked on the platform (`POST /admin/stocks/add`)
  - Update existing stock information (`PUT /admin/stocks/update`)
  - Remove stocks from the platform (`DELETE /admin/deleteStock/{stockId}`)

Building the Future: What's Next?

This platform is a strong foundation. Our modular, cloud-native architecture allows for exciting future enhancements.

Enhanced Analytics & Reporting:
Introduce advanced charts, performance attribution analysis, and downloadable portfolio reports.

AI-Powered Recommendations:
Develop a machine learning model to provide personalized investment suggestions based on user risk appetite and market trends.

CI/CD Pipeline Automation:
Implement a full CI/CD pipeline using AWS CodePipeline to automate testing and deployment for faster, more reliable updates.

Mobile-First Experience:
Develop native iOS and Android applications for on-the-go portfolio management.

Expand Asset Classes:
Integrate support for other financial instruments like ETFs, options, and cryptocurrencies.

A Comprehensive, Cloud-Native Financial Platform

We successfully designed, built, and deployed a full-stack stock portfolio management application that is **secure**, **scalable**, and **data-driven**.

By making deliberate architectural choices—like our **hybrid database model**, **JWT-based security**, and **serverless real-time updates**—we have created a robust platform that meets the demands of a modern financial application.

The project demonstrates a mastery of a diverse technology stack, from a **React** frontend to a **Spring Boot** backend, all powered by the flexibiity and strength of the **AWS cloud**.

Thank You

Questions? in IBM Plex Sans Regular