System Design

1. System Overview

The **Stock Portfolio Management System** is a web-based application designed for users to manage stock portfolios, view market trends, generate reports, and analyze predictions. The system also includes an admin module for managing users, stock data, and the machine learning model for stock predictions.

2. Architecture design

The system follows a **3-tier architecture**:

- **Presentation Layer**: User-facing web interface (frontend).
- **Application Layer**: Handles business logic and operations.
- **Database Layer**: Stores user data, stock information and portfolios.

2.1. Key Components

2.1.1. Frontend Layer

- Web Application (React.js)
- Mobile Application (React Native)
- Admin Dashboard (React.js)

2.1.2. API Gateway layer

- Authentication & Authorization
- Rate Limiting
- Request Routing
- Load Balancing

2.1.3. Microservices layer

- Authentication Service
- Portfolio Management Service
- Stock Data Service
- Prediction Service
- User Management Service
- Data layer
- Document Store (MongoDB)

2.2. Technology stack

2.2.1. Frontend

- Framework: React.js, Next.js
- Libraries: Chart.js
- Authentication: JWT json web token, Auth0 V2

2.2.2. Backend

- Framework: Node.js
- ML model: LSTM /Optimisation-sharpe ratio

2.2.3. Database

- MongoDB

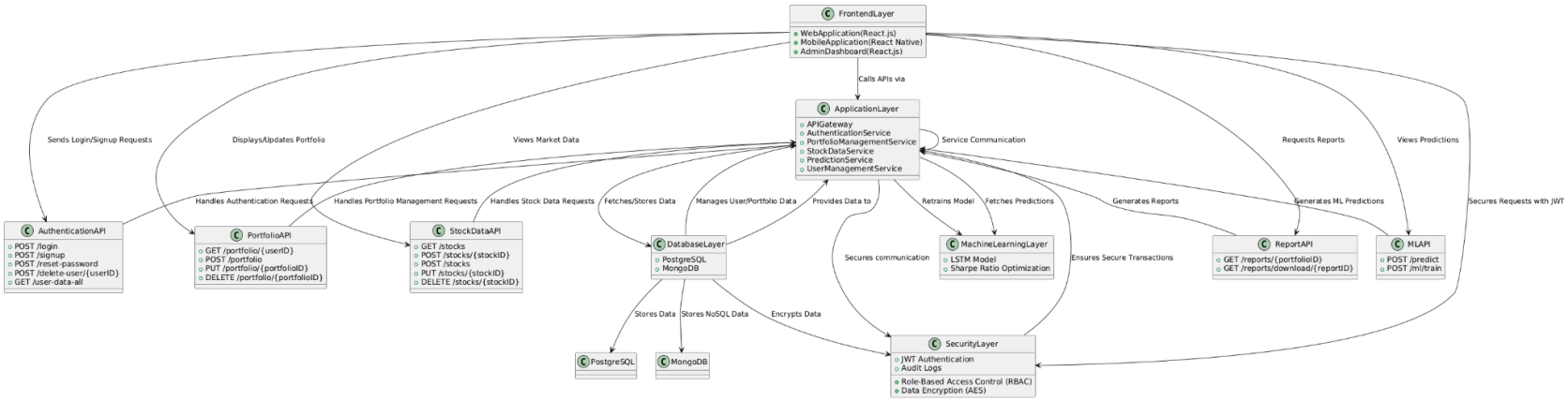
2.2.4. Testing

- Frontend testing:
  - GUI testing: Selenium
- Backend testing:
  - NF testing: JMeter, Blasze-Meter, Immuni Web, Uptime Robot
  - Unit testing: Mocha
  - API testing: Postman
- Database testing:
  - MongoDB
  - Atlas
  - Test cluster

2.2.5. Other tools

- Version control: Git/GitHub
- Deployment: Vercel
- IDE: Visual Studio Code

3. System Architecture diagram


FrontendLayer
WebApplication(React.js)
MobileApplication(React Native)
AdminDashboard(React.js)
Calls APIs via
ApplicationLayer
APIGateway
AuthenticationService
PortfolioManagementService
StockDataService
PredictionService
UserManagementService
Service Communication
Sends Login/Signup Requests
Displays/Updates Portfolio
Views Market Data
Requests Reports
Views Predictions
Handles Authentication Requests
Handles Portfolio Management Requests
Handles Stock Data Requests
Fetches/Stores Data
Manages User/Portfolio Data
Provides Data to
Retrains Model
Fetches Predictions
Generates Reports
Generates ML Predictions
Secures Requests with JWT
AuthenticationAPI
POST /login
POST /signup
POST /reset-password
POST /delete-user/{userID}
GET /user-data-all
PortfolioAPI
GET /portfolio/{userID}
POST /portfolio
PUT /portfolio/{portfolioID}
DELETE /portfolio/{portfolioID}
StockDataAPI
GET /stocks
POST /stocks/{stockID}
POST /stocks
PUT /stocks/{stockID}
DELETE /stocks/{stockID}
DatabaseLayer
PostgreSQL
MongoDB
MachineLearningLayer
LSTM Model
Sharpe Ratio Optimization
ReportAPI
GET /reports/{portfolioID}
GET /reports/download/{reportID}
MLAPI
POST /predict
POST /ml/train
Secures communication
Ensures Secure Transactions
Stores Data
Stores NoSQL Data
Encrypts Data
PostgreSQL
MongoDB
SecurityLayer
JWT Authentication
Audit Logs
Role-Based Access Control (RBAC)
Data Encryption (AES)

4. Functional modules

4.1. Authentication and User management

- Actor: User, Admin
- Features:
  - User signup, login, and password recovery.
  - Admin controls for managing user accounts.
- Flow:
  - The user enters credentials on the login page.
  - Backend authenticates using ()
  - Successful login grants a session token.

4.2. Portfolio management

- Actors: User
- Features:
  - View, add, update and delete portfolio stocks
  - Fetch performance reports
- Flow:
  - The user views their portfolio.
  - CRUD operations on portfolio data are handled via REST APIs.
  - Portfolio performance is calculated and visualized using charts.

4.3. Stock data management

- Actors: Users, Admin
- Features:
  - Users can view market trends and stock data.
  - Admins can add or edit stock data.
- Flow:
  - The system fetches real-time data via APIs (e.g., Alpha Vantage or Yahoo Finance).
  - Admin CRUD actions update stock details in the database.

4.4. Report management

- Actors: Users

- Features:
  - Users can generate portfolio reports.
- Flow:
  - Users trigger report generation.
  - Backend compiles data and displays it.

4.5. Machine learning predictions

- Actors: User, Admin
- Features:
  - Users access stock predictions based on historical data.
  - Admins update and retrain the ML model.
- Flow:
  - Historical data is fed into a predictive ML model.
  - Predictions are returned with confidence scores.
  - Admin retrains the model using new datasets.

5. Database design

5.1. Entities

5.1.1. Users table:

- UserID (PK)
- Name
- Email
- Password (hashed)
- CreatedAt
- UpdatedAt

5.1.2. Portfolio table:

- PortfolioID (PK)
- UserID (FK)
- PortfolioName
- CreatedAt
- UpdatedAt

5.1.3. Stocks table:

- StockID (PK)
- StockName
- Symbol
- Sector

- CurrentPrice
- LastUpdated

5.1.4. Transaction table:

- TransactionID (PK)
- PortfolioID (FK)
- StockID (FK)
- Type
- Quantity
- PurchasePrice
- TotalValue (Quantity * PurchasePrice)
- TransactionDate
- CreatedAT

5.2. Relationships

5.2.1. Relationship: User → Portfolio

- Type: One-to-Many
  - A User can have multiple Portfolios.
  - A Portfolio belongs to a single User.
- Implementation:
  - The userId field in the Portfolios collection acts as a foreign key referencing the _id field in the Users collection.

5.2.2. Relationship: Portfolio → Transaction

- Type: One-to-Many
  - A Portfolio can have multiple Transactions.
  - A Transaction belongs to a single Portfolio.
- Implementation:
  - The portfolioId field in the Transactions collection acts as a foreign key referencing the _id field in the Portfolios collection.

5.2.3. Relationship: Stock → Transaction

- Type: One-to-Many
  - A Stock can be part of multiple Transactions.
  - A Transaction is linked to a single Stock.
- Implementation:
  - The stockId field in the Transactions collection acts as a foreign key referencing the _id field in the Stocks collection.

5.2.4. Combined Relationship: User → Portfolio → Transaction → Stock

- A User owns Portfolios.

- Each Portfolio contains multiple Transactions.
- Each Transaction is linked to a specific Stock.

5.3. Entity-Relationship Mapping

5.3.1. Users Collection

- Primary Key: _id

5.3.2. Portfolios Collection

- Primary Key: _id
- Foreign Key: userId (links to Users → _id)

5.3.3. Transactions Collection

- Primary Key: _id
- Foreign Keys:
- portfolioId (links to Portfolios → _id)
- stockId (links to Stocks → _id)

5.3.4. Stocks Collection

- Primary Key: _id

6. API Design:

6.1. Authentication:

- POST /login: Authenticate user.
- POST /signup: Register a new user.
- POST /reset-password: Trigger password recovery.
- POST/delete-user/{userID}: Delete user and its details(Admin)
- GET/user-data-all: Get UserID and names of all users (Admin)

6.2. Portfolio management:

- GET /portfolio/{userID}: Fetch user portfolio.
- POST /portfolio: Add stocks to the portfolio.
- DELETE /portfolio/{portfolioID}: Remove stocks from the portfolio.

6.3. Stock data:

- GET /stocks: Fetch all stock data.
- POST/stocks/{stockID} - Fetch detailed information of a particular stock

6.4. Reports:

- GET /reports/{portfolioID}: Generate a report.

6.5. ML Predictions:

- POST /predict: Generate stock predictions

7. Security design

- **Authentication**: Use secure JWT tokens with role-based access control (RBAC).
- **Data Encryption**: Encrypt sensitive data like passwords and financial information using AES or similar standards.
- **API Security**: Protect APIs with rate-limiting, input validation, and HTTPS.
- **Audit Logs**: Record all actions performed by users and admins for compliance.

8. Non-functional requirements

- Performance
- Scalability
- Availability
- Security
- Reliability
- Usability
- Compliance
- Maintainability
- Interoperability