**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 analyse 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**:

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

**Technology Stack**

* **Frontend**:
  + Framework: React.js, Angular, or Vue.js.
  + Libraries: Chart.js or D3.js for visualizations.
  + Authentication: JWT for secure user sessions.
* **Backend**:
  + Framework: Node.js with Express.js, Django, or Spring Boot.
  + API: REST or GraphQL APIs for communication.
  + ML Framework: TensorFlow or PyTorch for stock prediction models.
* **Database**:
  + Relational DB: MySQL/PostgreSQL for structured data.
  + NoSQL DB: MongoDB for unstructured data (e.g., logs, reports).
* **Cloud Services** (Optional):
  + Hosting: AWS, Google Cloud, or Azure.
  + ML Deployment: AWS SageMaker or Google AI Platform.
* **Other Tools**:
  + GitHub for version control.
  + CI/CD Pipelines: Jenkins, GitHub Actions, or GitLab CI.

**3. Functional Modules**

**3.1 Authentication & User Management**

* **Actors**: 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 JWT or OAuth.
  + Successful login grants a session token.

**3.2 Portfolio Management**

* **Actors**: User.
* **Features**:
  + View, add, update, and delete portfolio stocks.
  + Fetch and download 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.

**3.3 Stock Data Management**

* **Actors**: User, Admin.
* **Features**:
  + Users can view market trends and stock data.
  + Admins can add, edit, or remove 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.

**3.4 Report Management**

* **Actors**: User.
* **Features**:
  + Users generate and download portfolio reports.
* **Flow**:
  + Users trigger report generation.
  + Backend compiles data into PDF or Excel format.
  + The report is provided as a downloadable file.

**3.5 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.

**4. Database Design**

**4.1 Entities**

1. **Users Table**:
   * UserID (PK)
   * Name
   * Email
   * Password (hashed)
   * Role (User/Admin)
   * CreatedAt
2. **Portfolio Table**:
   * PortfolioID (PK)
   * UserID (FK)
   * StockID (FK)
   * Quantity
   * CreatedAt
3. **Stocks Table**:
   * StockID (PK)
   * StockName
   * CurrentPrice
   * LastUpdated
4. **Reports Table**:
   * ReportID (PK)
   * PortfolioID (FK)
   * GeneratedAt
   * ReportFilePath
5. **ML Model Metadata**:
   * ModelID (PK)
   * ModelVersion
   * Accuracy
   * LastTrained

**5. API Design**

1. **Authentication**:
   * POST /login: Authenticate user.
   * POST /signup: Register a new user.
   * POST /reset-password: Trigger password recovery.
2. **Portfolio Management**:
   * GET /portfolio/{userID}: Fetch user portfolio.
   * POST /portfolio: Add stocks to the portfolio.
   * PUT /portfolio/{portfolioID}: Edit portfolio details.
   * DELETE /portfolio/{portfolioID}: Remove stocks from the portfolio.
3. **Stock Data**:
   * GET /stocks: Fetch all stock data.
   * POST /stocks: Add new stock data (Admin).
   * PUT /stocks/{stockID}: Update stock details (Admin).
   * DELETE /stocks/{stockID}: Remove stock data (Admin).
4. **Reports**:
   * GET /reports/{portfolioID}: Generate a report.
   * GET /reports/download/{reportID}: Download report.
5. **ML Predictions**:
   * POST /predict: Generate stock predictions.
   * POST /ml/train: Retrain the ML model (Admin).

**6. 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.

**7. Non-Functional Requirements**

1. **Scalability**: The system must handle increasing user loads and large volumes of stock data.
2. **Performance**: API responses should occur within 1 second for 90% of requests.
3. **Availability**: The system must be available 99.9% of the time.
4. **Security**: Ensure user data privacy and protection against threats like SQL injection and XSS.
5. **Usability**: The interface should be intuitive, with proper navigation and error handling.