For building a **Portfolio Creation and Management** feature in a **Stock Portfolio Management System**, here is a recommended technology stack. This stack is designed for scalability, security, and efficient handling of real-time data from stock market APIs.

**1. Frontend**

**Technologies**:

* **React.js**: For building interactive and dynamic user interfaces.
* **Redux**: For managing application state, especially useful for handling complex portfolio data.
* **Material UI** or **Bootstrap**: For pre-built components and consistent, responsive UI design.
* **Chart.js** or **D3.js**: For data visualization, such as portfolio performance graphs, pie charts, etc.

**Responsibilities**:

* Display real-time portfolio updates.
* Interact with backend services to create, edit, and view portfolios.
* Provide responsive and accessible UIs for managing stocks, creating visualizations, and confirming actions.

**2. Backend**

**Technologies**:

* **Node.js** with **Express.js**: For building a RESTful API to handle requests from the frontend.
* **Spring Boot** (Java alternative): A solid choice if the team prefers Java, for its robustness in building enterprise-level applications.
* **Python (Flask or Django)**: Also an alternative for its ease in data processing and analytics if the system requires extensive computational tasks.

**Responsibilities**:

* Handle CRUD operations for portfolio and stock management.
* Connect to external stock market APIs for real-time stock data retrieval.
* Perform backend calculations (e.g., portfolio value, ROI).
* Ensure secure authentication and authorization.

**3. Database**

**Technologies**:

* **PostgreSQL**: A relational database for storing structured portfolio, stock, and user data with robust querying capabilities.
* **MongoDB**: An alternative for handling complex and unstructured data, especially if additional metadata is needed for stocks.
* **Redis** (caching): To cache frequently requested data (like stock prices) and enhance response times.

**Responsibilities**:

* Persist user portfolios, stocks, and transaction history.
* Support transactional integrity (important for accurate portfolio updates).
* Provide optimized data access for analytics and reporting.

**4. Real-Time Data & Caching**

**Technologies**:

* **WebSocket** (e.g., using Socket.io): For real-time updates of stock prices, portfolio values, and alerts.
* **Redis** or **Memcached**: For caching stock price data and reducing API calls to stock market services.

**Responsibilities**:

* Provide real-time updates to users as stock prices change.
* Cache frequently requested stock data to reduce load on stock market APIs.

**5. External APIs**

**Technologies**:

* **Financial/Stock Market API** (e.g., Alpha Vantage, IEX Cloud, or Yahoo Finance API): To pull real-time and historical stock data for portfolio calculations.
* **Currency Exchange API** (optional): For users managing international portfolios in different currencies.

**Responsibilities**:

* Provide real-time data feeds for stock prices and market movements.
* Supply historical data for performance tracking and analysis.

**6. Authentication & Authorization**

**Technologies**:

* **JWT (JSON Web Token)**: For secure user authentication and session management.
* **OAuth2**: For handling third-party authentication if using social logins (e.g., Google, Facebook).
* **Spring Security** (if using Spring Boot): Provides advanced security options for role-based access control.

**Responsibilities**:

* Authenticate users and authorize access to portfolio data.
* Secure sensitive data, especially when interacting with stock data APIs and user financial data.