**Requirement Gathering and Analysis Phase**

**Solution Architecture**

|  |  |
| --- | --- |
| Date |  |
| Team ID | SWTID1720195916 |
| Project Name | Stock Trading Web App |
| Maximum Marks |  |

**Solution Architecture:**

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

* Find the best tech solution to solve existing business problems.
* Describe the structure, characteristics, behavior, and other aspects of the software to project stakeholders.
* Define features, development phases, and solution requirements.
* Provide specifications according to which the solution is defined, managed, and delivered.

**Solution Architecture of Project:**

1. **Client:**

* User: Represents the end-user who interacts with the stock trading web app.
* Stock Trading Web App: The front-end application accessed by the user through their browser.

1. **Browser:** The web browser used by the client to access the web application.
2. **WebServer:**

* Node.js: The JavaScript runtime environment used for building the server-side of the application.
* Express.js: A web application framework for Node.js, used to handle HTTP requests and route them to the appropriate handlers.

1. **AppServer**:

* React.js: The JavaScript library used for building the user interface of the web app. React components are rendered in the browser, providing a dynamic and responsive user experience.

1. **Database**:

* MongoDB: The NoSQL database used to store user information, stock data, transaction history, and other relevant data for the web app.

1. **ExternalServices**:

* Stock Market Data API: An external service used to fetch real-time stock market data, which is crucial for providing up-to-date information to the users.
* Notification Service: An external service used to send notifications to users about their transactions, stock price changes, or other important events.

**Interaction Flow:**

1. **Client Interaction:**

* The user interacts with the stock trading web app through their browser.

1. **Request Flow:**

* The browser sends an HTTP request to the WebServer (Node.js with Express.js) to access the web app.

1. **User Actions:**

* The user interacts with the React components in the app (e.g., checking stock prices, making transactions).
* These interactions result in API calls from the React app to the WebServer.

1. **Data Operations:**

* The WebServer handles these API calls and performs CRUD (Create, Read, Update, Delete) operations on the MongoDB database.

1. **External Services Interaction:**

* The WebServer fetches real-time stock data from the Stock Market Data API and sends notifications via the Notification Service when necessary.
* The fetched data is saved in the MongoDB database for future use and quick access.

1. **Response Flow:**

* The WebServer processes the data and returns the API responses to the React app.
* The React app updates the user interface based on the responses, providing the user with real-time data and feedback.

**Notes:**

* Client Note: The client interacts with the React.js application, which is served by the Node.js and Express.js backend.
* Database Note: MongoDB is used for storing user information, stock data, and transaction history.
* External Services Note: External services are used for fetching real-time stock market data and sending notifications to users.

**Solution Architecture Diagram:**

A diagram of a software application

Description automatically generated with medium confidence

Figure 1: Architecture and data flow of the stock trading web application