COLLEGE CODE: 8203

COLLEGE: AVC COLLEGE OF ENGINEERING

DEPARTMENT: INFORMATION TECHNOLOGY

STUDENT NM-ID: 4D2198FA90720AD8F834BBD4C69121C9

ROLL NO: 23IT100

DATE:22-09-2025

Completed the project named as Phase 3

TECHNOLOGY PROJECT NAME: Admin dashboard with charts

SUBMITTED BY,

NAME: Sivarajaganapathi S

MOBILE NO: 7845102808

1. Project Setup

This phase establishes the foundational environment for both the backend API and the frontend dashboard.

Component	Action	Tools/Dependencies
Backend Setup	Initialize Node.js project. Create the main server file (server.js) and configure the Express server.	**Node.js, Express.js**
Frontend Setup	Initialize the React application shell (e.g., using Vite or Create React App).	**React**
Database Connection	Configure Mongoose to connect the Node.js server to the MongoDB database instance.	**MongoDB, Mongoose**
Dependency Install	Install core libraries for security, data fetching, and visualization on both sides.	**express, mongoose, jsonwebtoken (JWT), axios, react, chart.js, react-chartjs-2**
Configuration	Set up environment variables (.env) for the MongoDB URI, JWT secret key, and server port.	**dotenv**

2. Core Features Implementation

This phase involves building the essential functionality of the dashboard, focusing on data security and visualization.

Backend (Node.js/Express/MongoDB)

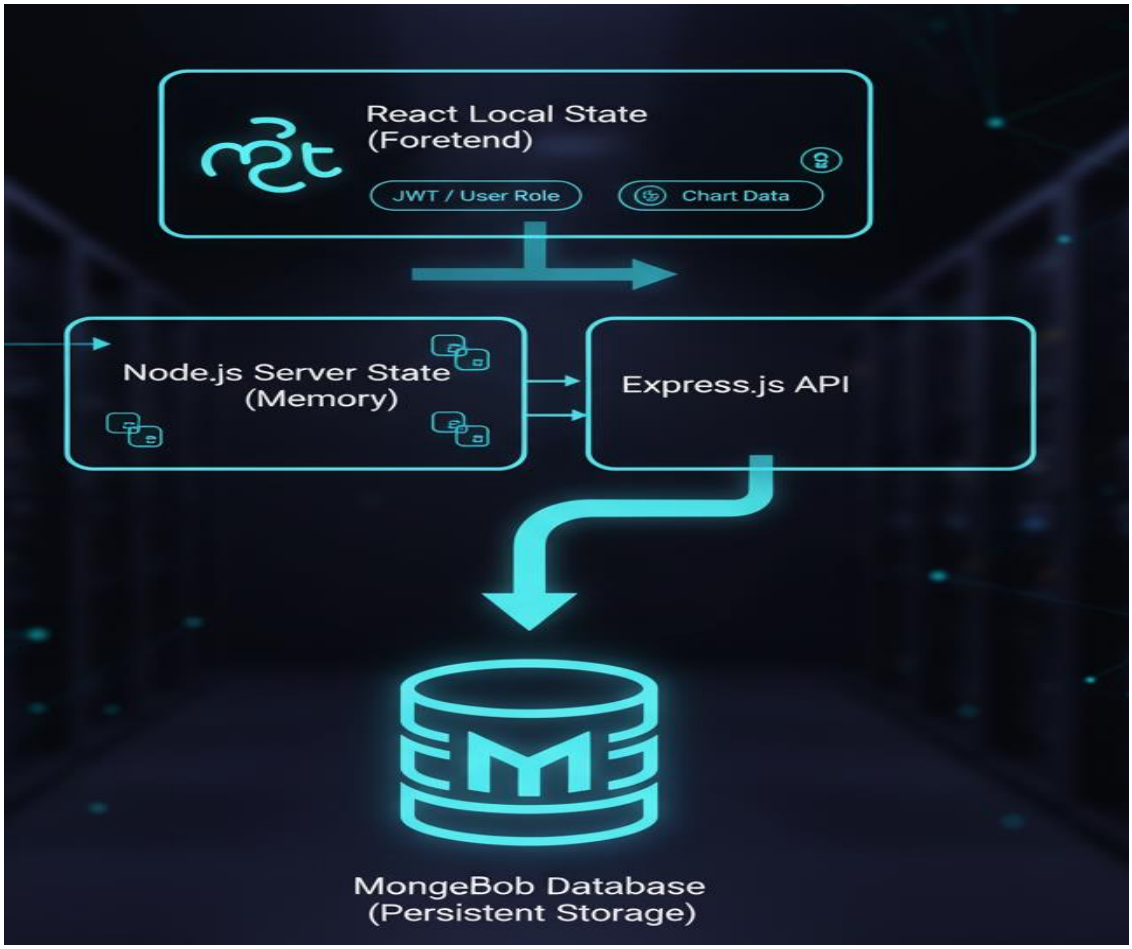
1. **Metric Computation Logic:** Implement specialized **MongoDB aggregation pipelines** to efficiently calculate complex metrics like daily sales totals, user growth over time, and traffic source distribution.
2. **Authentication Endpoint:** Create the /api/auth/login endpoint to validate user credentials and return a **JSON Web Token (JWT)** containing the user's role.
3. **Role-Based Access Control (RBAC) Middleware:** Develop middleware that runs on every metric route (/api/metrics/*). This middleware verifies the JWT and checks if the embedded user **role** is authorized to access the specific data.
4. **Metric API Endpoints:** Create secured GET endpoints (e.g., /api/metrics/sales, /api/metrics/users) that execute the aggregation pipelines and send the computed metric data to the frontend in a clean **JSON format** ({ labels: [...], data: [...] }).

Frontend (React/Chart.js)

1. **Secure Data Fetching:** Use **Axios** to send requests to the backend APIs, including the JWT in the Authorization header.
2. **Chart Visualization:** Utilize **react-chartjs-2** wrappers to integrate **Chart.js**. Create reusable components for **Line Charts** (e.g., User Signups Over 30 Days), **Bar Charts** (e.g., Sales by Product Category), and **Pie Charts** (e.g., Traffic Source).
3. **Chart Auto-Update:** Implement the useEffect hook with setInterval to periodically refetch data from the metric APIs (e.g., every 30 seconds) and update the chart data state, achieving a near real-time feel.
4. **Frontend RBAC UI Logic:** Use conditional rendering in React to show or hide entire chart components or metric cards based on the authenticated user's **role** stored in the local state.

3. Data Storage (Local State / Database)

Storage Layer	Purpose	Content	Technology
Database (Persistent)	Stores all application and transactional data needed for metric calculation.	User Accounts, Sales Records, View Logs, System Events.	**MongoDB**
Local State (Frontend)	Stores runtime data critical for application function and security.	**JWT/User Role** (stored after login), **Fetched Metric Data** (labels and values for charts).	**React State Management** (e.g., useState, Context API)
Server State (Memory)	Temporarily holds the results of metric calculations before sending the JSON response.	Computed metric arrays/objects.	**Node.js/Express**



4. Testing Core Features

Rigorous testing is essential, particularly for the security and data accuracy features.

Test Type	Objective	Tools & Methods	Test Case Example
Unit Testing	Verify individual function logic (security and computation).	Jest/Mocha (for Node.js), Mocks	Test the calculateSales function to ensure it returns the correct sum for a given date range.
Integration Testing	Verify API endpoints and security middleware interaction.	**Postman** / Supertest	Attempt to access /api/metrics/sales with no JWT → **Assert 401 Unauthorized**. Attempt with a 'Manager' JWT → **Assert 403 Forbidden**.
End-to-End (E2E) Testing	Verify the full user flow and auto-update mechanism.	Manual Testing (Browser)	Log in as an 'Admin' user and verify that all charts load and refresh after 30 seconds. Log in as a 'Basic' user and verify the 'Sales' chart is **hidden**.
Data Accuracy	Ensure charts reflect the data retrieved from MongoDB.	Console Logging, Database Checks	Manually query MongoDB for a metric, then compare the result to the value displayed on the corresponding Chart.js component.

5. Version Control (GitHub)

1. **Initialization:** Create a new remote repository on **GitHub** and initialize the local project as a Git repository.
2. **Configuration:** Create a .gitignore file to exclude sensitive information and unnecessary files (e.g., node_modules/, .env).

3. **Branching Strategy:** Use a clear branching model (e.g., **main** for production code, **develop** for stable feature integration, and short-lived **feature branches** for individual tasks).
4. **Workflow:** Commit changes frequently with descriptive messages linked to the features being implemented (e.g., "FE: Implemented Chart.js Line Chart component," "BE: Added RBAC middleware for metric endpoints").
5. **Collaboration:** Regularly push and pull changes to ensure the team is working on the latest codebase.
6. **https://github.com/sivarajaganapathi12/NM-phase3.git**