COLLEGE CODE : 9111

COLLEGE NAME : SRM Madurai College For Engineering And Technology

DEPARTMENT: Computer Science And Engineering

STUDENT NM-ID : 1AB1AA81B6B40D9B5C94D469B460D04A

1FFEB20A6540161A1F54E258167D1978

877B09181AE2E069AFF4C36AF16D1C4D

FF192F92083F64EB3754B0596244F018

ROLL NO ; 911123104040

911123104016

911123104032

911123104004

DATE : 15/09/2025

Completed the project named as phase 3

TECHNOLOGY PROJECT NAME : REAL TIME STOCK TICKER

SUBMITTED BY:

Rajesh Kumar M	mobile no : 8524925826
Harish manikandan R	mobile no : 6369119261
Nyson DM	mobile no : 8682054707
Arul kumaran M	mobile no : 8428853991

1. Project Setup

A well-structured project setup ensures scalability, collaboration, and maintainability

Environment Setu

- **Backend:** Install Node.js, Express, and WebSocket libraries.
- **Frontend:** React.js + Next.js for SSR (server-side rendering).
- **Database:** PostgreSQL, Redis, and TimescaleDB.
- **Dev Tools:** Docker for containerization, ESLint + Prettier for linting, Postman for API testing.

```
# Backend setup
mkdir realtime-stock-ticker && cd realtime-stock-ticker
npm init -y
npm install express socket.io cors dotenv

# Frontend setup
npx create-next-app frontend
cd frontend
npm install axios recharts socket.io-client tailwindcss
```

Project Structure

```
/realtime-stock-ticker
   /backend
      server.js
      routes/
      models/
   /frontend
      pages/
      components/
      utils/
   /database
      schema.sql
   /tests
```

---

2. Core Features Implementation

1. Real-Time Stock Updates

- Use WebSocket to stream stock data.
- Display updates instantly on frontend.

2. Stock Watchlist

- Users can add/remove stocks.
- Persisted in DB with user authentication.

3. Alerts & Notifications

- Users define thresholds.
- Backend triggers events when stock crosses limit.

Real-Time Stock Updates

```
// backend/server.js
const io = require("socket.io")(3001, { cors: { origin: "*" }
});
setInterval(() => {
  const stockData = { symbol: "AAPL", price: (150 +
Math.random() * 5).toFixed(2) };
  io.emit("stockUpdate", stockData);
}, 2000);
```

Watchlist Management

```
// frontend/utils/watchlist.js
let watchlist = [];
export const addToWatchlist = (symbol) => {
  if (!watchlist.includes(symbol)) watchlist.push(symbol);
};
export const removeFromWatchlist = (symbol) => {
  watchlist = watchlist.filter((s) => s !== symbol);
};
```

Alerts

```
// backend/routes/alerts.js
app.post("/alerts", (req, res) => {
  const { symbol, targetPrice } = req.body;
  alerts.push({ symbol, targetPrice });
  res.json({ msg: "Alert created" });
});
```

3. Data Storage (Local State / Database)

1. Local State Management

- Use **React Context API or Redux**.
- Stores session data (current stock list, filters).

2. Data Handling Approach

- **Frontend State**: Fast, temporary storage.
- **Database**: Persistent watchlists, user accounts.
- **Caching Layer**: Redis for high-frequency queries

Frontend Local State (React Context)

```
import { createContext, useState } from "react";
export const StockContext = createContext();
export function StockProvider({ children }) {
  const [stocks, setStocks] = useState([]);
  return <StockContext.Provider value={{ stocks, setStocks
}}>{children}</StockContext.Provider>;
}
```

Backend Database Models

```
-- database/schema.sql
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  username VARCHAR(100),
  password VARCHAR(255)
);

CREATE TABLE watchlist (
  id SERIAL PRIMARY KEY,
  user_id INT REFERENCES users(id),
  symbol VARCHAR(10)
);

CREATE TABLE alerts (
  id SERIAL PRIMARY KEY,
  user_id INT REFERENCES users(id),
  symbol VARCHAR(10),
  target_price NUMERIC
);
```

4. Testing Core Features

1. Unit Testing

- Test API endpoints with Jest.
- Validate data parsing and formatting.

2. Integration Testing

- Verify DB + API interactions.
- Ensure WebSocket updates are broadcast.

3. End-to-End Testing

- Cypress tests user flows:
    - Login → Add stock → View updates.

4. Load Testing

- Simulate multiple users subscribing to stock updates.
- Use **k6** or **Apache JMeter**.

Unit Testing (Jest)

```
// backend/tests/stock.test.js
const request = require("supertest");
const app = require("../server");

test("GET stock price", async () => {
  const res = await request(app).get("/stocks/AAPL");
  expect(res.statusCode).toBe(200);
  expect(res.body).toHaveProperty("symbol");
});
```

E2E Testing (Cypress)

```
// frontend/cypress/integration/stock_spec.js
describe("Stock Dashboard", () => {
  it("displays real-time stock updates", () => {
    cy.visit("http://localhost:3000");
    cy.contains("AAPL");
  });
});
```

---

5. Version Control (GitHub)

Repository Setup

- Create GitHub repo.
- Setup `.gitignore` for node_modules and environment files.

2. Branching Strategy

- **main** → stable production.
- **dev** → ongoing development.
- **feature/** → per feature branches.

3. Commit Guidelines

  - `feat: add stock alert system`
  - `fix: resolve WebSocket disconnect bug`

4. Pull Request Workflow

- All PRs must be reviewed before merging.
- Use GitHub Actions to run tests on PR.

Branching Strategy

- `main` → production-ready.
- `dev` → staging and integration.
- `feature/*` → new features.
- `bugfix/*` → fixes.

```
# Initialize git
git init
git remote add origin https://github.com/user/realtime-stock-
ticker.git

# Create branches
git checkout -b dev
git checkout -b feature/watchlist
```

GitHub Actions (CI/CD)

```
# .github/workflows/node.js.yml
name: Node.js CI
on: [push]
```

```
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Use Node.js
        uses: actions/setup-node@v2
        with:
          node-version: "18"
      - run: npm install
      - run: npm test
```