Python

```
import asyncio

import aiohttp

import json

import sqlite3

import random

from time import time

import matplotlib.pyplot as plt

import numpy as np


# Define the Stock API URL (hypothetical API endpoint)

API_URL = "https://api.stockmarket.com/v1/stocks/{symbol}/price"


# Simulate API key

API_KEY = "your_api_key_here"


# Retry mechanism with exponential backoff

async def fetch_with_retry(url, headers, session, retries=3, backoff=1):

    """Fetch data with retries and exponential backoff."""

    attempt = 0

    while attempt < retries:

        try:

            async with session.get(url, headers=headers) as response:

                if response.status == 200:
```

```
                return await response.json()
            else:
                raise Exception(f"Failed with status {response.status}")
        except Exception as e:
            attempt += 1
            print(f"Attempt {attempt} failed: {e}")
            await asyncio.sleep(backoff * (2 ** attempt))  # Exponential backoff
    raise Exception("Max retries exceeded")

# Function to fetch stock data asynchronously with retry
async def fetch_stock_data(symbol: str, session: aiohttp.ClientSession):
    """Fetches stock data for a given symbol asynchronously with retries."""
    url = API_URL.format(symbol=symbol)
    headers = {"Authorization": f"Bearer {API_KEY}"}
    data = await fetch_with_retry(url, headers, session)
    return data['price']  # Return the price from the API response

# Asynchronous function to process multiple stock symbols concurrently
async def fetch_multiple_stocks(symbols: list):
    """Fetches stock data for multiple symbols concurrently."""
    async with aiohttp.ClientSession() as session:
        tasks = [fetch_stock_data(symbol, session) for symbol in symbols]
        results = await asyncio.gather(*tasks)
        return results
```

```
# Calculate the average stock price from the results
def calculate_average_price(prices):
    """Calculate the average stock price."""
    return sum(prices) / len(prices) if prices else 0


# Function to save the financial report in an SQLite database
def save_report_to_db(report):
    """Saves the financial report to an SQLite database."""
    conn = sqlite3.connect("financial_reports.db")
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS reports (
                        report_generated_at REAL,
                        execution_time REAL,
                        average_price REAL)''')
    cursor.execute("INSERT INTO reports (report_generated_at, execution_time, average_price)
VALUES (?, ?, ?)",
                   (report['report_generated_at'], report['execution_time'], report['average_price']))
    conn.commit()
    conn.close()


# Function to generate a financial report
async def generate_financial_report(symbols: list):
    """Generates a financial report based on stock price data."""
    start_time = time()
    prices = await fetch_multiple_stocks(symbols)
    avg_price = calculate_average_price(prices)
    report = {
        "symbols": symbols,
        "average_price": avg_price,
        "stock_prices": prices,
        "report_generated_at": time(),
        "execution_time": round(time() - start_time, 2)
    }

    # Save the report in the database
    save_report_to_db(report)

    # Visualize the stock prices using matplotlib
    visualize_stock_data(symbols, prices)

    return report

# Function to visualize stock data (basic line graph)
def visualize_stock_data(symbols, prices):
    """Visualizes stock data using matplotlib."""
    plt.figure(figsize=(10, 5))
    plt.plot(symbols, prices, marker='o', linestyle='-', color='b')
    plt.title('Stock Prices for Selected Symbols')
    plt.xlabel('Stock Symbols')
    plt.ylabel('Stock Price')
    plt.grid(True)
    plt.show()

# Main entry point for the script
async def main():
    # Define stock symbols to track
    stock_symbols = ["AAPL", "GOOGL", "AMZN", "MSFT", "META"]

    # Generate and print the financial report
    report = await generate_financial_report(stock_symbols)

    # Display the report
    print("Financial Report Generated:")
    print(json.dumps(report, indent=4))

# Running the script
if __name__ == "__main__":
    asyncio.run(main())
```