Assignment 4: Developing an Analytical Platform Using Node.js and MongoDB

Objective

Build an analytical platform to retrieve time-series data from a MongoDB database and visualize it using graphs. The platform should allow users to:

- Filter data by date range.
- Select specific fields for analysis.
- View metrics like average, minimum, maximum, and standard deviation.

Requirements

1. MongoDB Schema Design

- Create a measurements collection in MongoDB.
- Each document should include:
    - timestamp (Date): Time the measurement was recorded.
    - field1 (Number): Example - temperature, sales, stock prices.
    - field2 (Number): Example - humidity, customer satisfaction, product rating.
    - field3 (Number): Example - CO2 levels, units sold, website traffic.
    - Additional fields can be added based on your use case.

2. Backend Development (Node.js/Express)

- Set up a **Node.js** project and install dependencies:
  npm init -y npm install express mongoose
- Create an index.js file to set up an Express application and connect to MongoDB using Mongoose.
- Develop RESTful API endpoints to:
    - **Fetch Time-Series Data**: Retrieve data from the measurements collection filtering by start_date and end_date.
    - **Fetch Metrics**: Calculate and return metrics such as average, minimum, maximum, and standard deviation for the selected field.

Example Endpoints:

- GET /api/measurements?field=field1&start_date=YYYY-MM-DD&end_date=YYYY-MM-DD
  Response: [
  { "timestamp": "2025-01-01T12:00:00Z", "field1": 22.5 },
  { "timestamp": "2025-01-01T13:00:00Z", "field1": 23.1 }

]

- GET /api/measurements/metrics?field=field2
  Response: {
  "avg": 22.8,
  "min": 22.5,
  "max": 23.1,
  "stdDev": 0.3
  }

4. Frontend Development

- Create a basic frontend using HTML and JavaScript.
- Use libraries like **Chart.js** or **D3.js** to visualize data in graphs (e.g., line charts, bar charts).
- Provide a user-friendly interface to:
  - Select fields to visualize by input time ranges for filtering data.
  - Display time-series graphs and statistical metrics.

5. Visualization and Metrics

Ensure the platform supports:

- **Visualization**:
  - Time-series graphs for selected fields.
- **Metrics**:
  - Average value over the selected time range.
  - Minimum and maximum values.
  - Standard deviation.

6. Error Handling and Validation

- Implement robust error handling, including:
  - Invalid field names or date formats.
  - Missing data in the specified range.
- Validate inputs to prevent crashes and return meaningful error messages.

Additional Suggestions

Data Sources

You may use free datasets to populate your MongoDB database with *mongoimport* in CSV or JSON format. Examples include:

- **Kaggle**: E-commerce, finance, or weather data.

- **Data.gov**: U.S. crime or healthcare statistics.
- **OpenWeatherMap API**: Real-time and historical weather data.
- **World Bank**: Global economic and population data.

Submission Guidelines

- Submit the following:
    - A .zip file or a GitHub repository link containing your project.

Grading Criteria:

Criteria	Description	Weight
Backend Implementation	Efficiently develops backend functionality using Node.js and Express.js to interact with MongoDB.	20%
Database Design	Creates a well-structured schema for time-series data in MongoDB, ensuring scalability and clarity.	10%
API Functionality	Implements robust RESTful API endpoints for fetching, filtering, and retrieving statistical metrics.	20%
Frontend Usability	Builds an interactive and user-friendly frontend with clear data visualization (e.g., graphs, charts).	20%
Error Handling	Handles invalid inputs gracefully and ensures data integrity.	15%
Presentation & Defense	Effectively explains the design choices, challenges, and functionality during the project defense.	15%