**Task: API Development and MongoDB Integration**

**Objective:**

Create a system that allows uploading of stock data (in CSV format), validates the data, stores it in a MongoDB database, and performs specific calculations accessible through an API, using **Node.js**.

URL for getting Stock DATA - <https://www.kaggle.com/datasets/rohanrao/nifty50-stock-market-data>

**It can be downloaded for free.**

**Part 1: CSV Upload and Data Validation**

1. **API Endpoint:**
   * Create a **POST** endpoint (/upload) using **Express.js** that accepts a CSV file.
2. **File Format Verification:**
   * Ensure that the uploaded file is in CSV format.
   * Verify that the following columns are present in the CSV file:
     + Date, Symbol, Series, Prev Close, Open, High, Low, Last, Close, VWAP, Volume, Turnover, Trades, Deliverable, %Deliverable
3. **Data Validation:**
   * Check each row for the following conditions:
     + Date: Should be in a valid date format.
     + Prev Close, Open, High, Low, Last, Close, VWAP, Volume, Turnover, Trades, Deliverable, %Deliverable: Should be numerical values.
   * If a row fails validation, log the row and skip it.
   * Keep track of the number of successful and failed records during the upload.
4. **Response:**
   * After processing the file, return a JSON response that indicates:
     + Total number of records in the file.
     + Number of successfully uploaded records.
     + Number of failed records with reasons for failure.

**Part 2: MongoDB Integration**

1. **Database Connection:**
   * Use the **MongoDB Node.js Driver** or **Mongoose** to connect to a MongoDB database and create a collection called stock\_data.
   * Each valid row from the CSV file should be inserted into this collection.
2. **Data Storage Structure:**
   * Store each record as a document in MongoDB with the following structure:

json

Copy code

{

"date": "YYYY-MM-DD",

"symbol": "ULTRACEMCO",

"series": "EQ",

"prev\_close": 10.0,

"open": 305.0,

"high": 340.0,

"low": 253.25,

"last": 259.0,

"close": 260.0,

"vwap": 268.8,

"volume": 6633956,

"turnover": 1.78E14,

"trades": 133456,

"deliverable": 970249,

"percent\_deliverable": 0.1463

}

**Part 3: Calculations and Data Retrieval**

1. **Get APIs:**
   * Implement the following **GET** APIs using **Express.js**. Each API should accept query parameters to filter by date range or symbol:
   * **API 1:**
     + **Endpoint:** /api/highest\_volume
     + **Parameters:**
       - start\_date, end\_date (e.g., ?start\_date=2024-01-01&end\_date=2024-12-31)
       - symbol (optional, e.g., ?symbol=ULTRACEMCO)
     + **Description:** Return the record(s) with the highest volume within the specified date range or for the specified symbol.
   * **API 2:**
     + **Endpoint:** /api/average\_close
     + **Parameters:**
       - start\_date, end\_date (e.g., ?start\_date=2024-01-01&end\_date=2024-12-31)
       - symbol (e.g., ?symbol=ULTRACEMCO)
     + **Description:** Calculate and return the average closing price (close field) within the specified date range for the specified symbol.
   * **API 3:**
     + **Endpoint:** /api/average\_vwap
     + **Parameters:**
       - start\_date, end\_date (e.g., ?start\_date=2024-01-01&end\_date=2024-12-31)
       - symbol (optional, e.g., ?symbol=ULTRACEMCO)
     + **Description:** Calculate and return the average VWAP within the specified date range or for the specified symbol.
2. **Query Examples:**
   * /api/highest\_volume?start\_date=2024-01-01&end\_date=2024-12-31
   * /api/average\_close?start\_date=2024-01-01&end\_date=2024-12-31&symbol=ULTRACEMCO
   * /api/average\_vwap?symbol=ULTRACEMCO
3. **Response Format:**
   * Return the data in JSON format, e.g.,

json

Copy code

{

"highest\_volume": {

"date": "YYYY-MM-DD",

"symbol": "ULTRACEMCO",

"volume": 1000000

}

}

* + For average APIs:

json

Copy code

{

"average\_close": 265.75

}

**Part 4: Documentation and Testing**

1. **Documentation:**
   * Provide clear documentation for each API, describing the parameters, expected input, and response format.
2. **Postman Collection:**
   * Create a Postman collection for all APIs, with pre-configured requests that can be used to test the system.
   * Include instructions on how to use the collection and test each API endpoint.
3. **Unit Tests:**
   * Write unit tests using **Jest** or **Mocha** for the validation logic and the calculations.

**Evaluation Criteria:**

* **Correctness:** All features work as expected.
* **Code Quality:** Code is clean, modular, and well-documented.
* **Error Handling:** System gracefully handles errors and edge cases.
* **Efficiency:** Implementation is efficient in terms of data handling and calculations.
* **Testing:** Well-written unit tests and a functional Postman collection.

**.** you need to host it.

**. Deadline: 24 hours**