Assignment 8
Group 38
Jake Back, Harry Cho
https://github.com/backhs97/Group38/tree/main

Building an End-to-End IoT System

IoT System End-to-End Architecture Approach

The system designed in this project combines the use of a TCP client-server architecture, MongoDB for database management, and metadata provided by Dataniz to ensure flexibility and increased functionality in the system. The TCP client enables user interaction with the system via a menu-driven interface. Users can ask queries against the IoT system. These queries are then processed at the server, which retrieves data from MongoDB, does any necessary calculations, and sends results back in a user-friendly format.

The server uses a binary search tree for efficient data organization and retrieval, making it most suitable for handling time-series IoT data. In that, metadata such as device names and timestamps are widely utilized to facilitate query processing. Calculations like RH% and unit conversions (e.g., gallons, kWh) are conducted to meet the requirements of the assignment in terms of accuracy and relevance of data.

Research Findings on IoT Sensors and Data

The IoT system generates data using three types of sensors:

1. Moisture Sensor: This sensor provides the measurement of moisture in the kitchen fridge. The data is captured in raw values and transformed into RH% using a linear mapping formula.
2. Water Flow Sensor: This sensor measures the amount of water used per cycle in the smart dishwasher. The data comes in gallons, as required by the imperial unit.
3. Electricity Consumption Sensor: Monitors power usage of three IoT devices (two refrigerators and one dishwasher). Data is stored in kWh and analyzed to identify the highest-consuming device.

```
{
  _id: ObjectId('6750dabbd50fad3a89afa958'),
  cmd: 'publish',
  retain: false,
  qos: 0,
  dup: false,
  length: 428,
  payload: {
    timestamp: '1733352123',
    topic: 'metadata',
    parent_asset_uid: '08e06c94-246e-49f0-80fa-166efa1a8e8b',
    asset_uid: '4a5fe66e-26b5-4f9a-a064-6bdde173e446',
    board_name: 'board 1 08e06c94-246e-49f0-80fa-166efa1a8e8b',
    'sensor 1 08e06c94-246e-49f0-80fa-166efa1a8e8b': '4.1839',
    'sensor 2 08e06c94-246e-49f0-80fa-166efa1a8e8b': '38.6286',
    'sensor 3 08e06c94-246e-49f0-80fa-166efa1a8e8b': '6.0894'
  },
  topic: 'metadata',
  time: 2024-12-04T22:42:03.000Z,
  __v: 0
}
```

```
{
  _id: ObjectId('6750db00d50fad3a89afab0c'),
  cmd: 'publish',
  retain: false,
  qos: 0,
  dup: false,
  length: 276,
  payload: {
    timestamp: '1733352192',
    topic: 'metadata',
    parent_asset_uid: '3m9-47q-7nk-3kn',
    asset_uid: '109-ptx-9ow-52p',
    board_name: 'Arduino Pro Mini - boardfridge',
    thermistor: '4.8586',
    'Moisture Meter - moist': '0.0000',
    ammeter: '25.6050'
  },
  topic: 'metadata',
  time: 2024-12-04T22:43:12.000Z,
  __v: 0
}
```

```
{
  _id: ObjectId('6750db02d50fad3a89afab7c'),
  cmd: 'publish',
  retain: false,
  qos: 0,
  dup: false,
  length: 252,
  payload: {
    timestamp: '1733352194',
    topic: 'metadata',
    parent_asset_uid: 'k3p-mwn-z4x-398',
    asset_uid: '075-u77-md0-3tl',
    board_name: 'Raspberry Pi 4 - new smart refrigerator',
    watercon: '55.1909',
    'dish ammeter': '4.3347'
  },
  topic: 'metadata',
  time: 2024-12-04T22:43:14.000Z,
  __v: 0
}
```


evi...	Sensor	Timestamp	Topic	Value
e...	parent_asset_uid	12/7/2024, 5:51:04 PM	metad...	08e06...
e...	asset_uid	12/7/2024, 5:51:04 PM	metad...	4a5fe6...
e...	sensor 1 08e06c94...	12/7/2024, 5:51:04 PM	metad...	3.9449
e...	sensor 2 08e06c94...	12/7/2024, 5:51:04 PM	metad...	24.3922
e...	sensor 3 08e06c94...	12/7/2024, 5:51:04 PM	metad...	1.8383

These sensors provide high-precision data and use UTC timestamps, which are converted to PST for user queries. Metadata like device names, timestamps, and sensor-specific fields enhances query processing and supports unit conversions.

Metadata Usage

Metadata is an important part of the system's functionality provided by Dataniz. Every IoT device has metadata that includes:

1. Timestamps: This will help in filtering data within a certain time range, like in the last three hours.
2. Device Names: It helps to identify specific devices, like "Smart Dishwasher", to query.

3. Sensor Data Types: It helps process the data correctly, such as moisture, water flow, and electricity consumption.

The metadata ensures that queries are processed with precision and flexibility. For example, the server uses metadata to distinguish between devices when calculating average electricity consumption. The system's reliance on metadata underlines its importance for query accuracy and scalability.

Algorithms, Calculations, and Unit Conversions

The server employs several algorithms and calculations to ensure efficient query processing and meaningful data presentation. A binary search tree was adopted to sort the IoT data in the memory for quick sorting and searching based on timestamps, which is highly effective in time-series data queries such as records from the last three hours. Calculations such as converting moisture level into RH% use a formula with linear interpolation. This mapping provides a translation of raw sensor data into a user-friendly format by establishing minimum and maximum values for temperature and RH%. Further, unit conversions are also part of the system. Water consumption data is translated into gallons, and electricity usage is in kWh, with both following the requirement to use imperial units. Timezone adjustments are made using the pytz library to convert timestamps to UTC, ensuring the results are aligned with the specified time zone. These algorithms and conversions work in harmony to process complex queries while maintaining accuracy and relevance.

```python
class BinarySearchTree:
    def __init__(self):
        self.root = None

    #insert a new node into the binary tree done recursively
    def insert(self, data, key="payload.timestamp"):
        if not self.root:
            self.root = TreeNode(data)
        else:
            self._insert_recursive(self.root, data, key)

    def _insert_recursive(self, node, data, key):
        if data.get(key) < node.data.get(key):
            if node.left is None:
                node.left = TreeNode(data)
            else:
                self._insert_recursive(node.left, data, key)
        else:
            if node.right is None:
                node.right = TreeNode(data)
            else:
                self._insert_recursive(node.right, data, key)

    #Search for a node with a specific key done recursively
    def search(self, key_value, key="payload.timestamp"):
        return self._search_recursive(self.root, key_value, key)

    def _search_recursive(self, node, key_value, key):
        if node is None or node.data[key] == key_value:
            return node
        elif key_value < node.data[key]:
            return self._search_recursive(node.left, key_value, key)
        else:
            return self._search_recursive(node.right, key_value, key)
```

```python
    #in-order traversal to retrieve sorted data
    def inorder_traversal(self, node=None, results=None):
        if self.root is None:
            return []
        if results is None:
            results = []
        if node is None:
            node = self.root
        if node.left:
            self.inorder_traversal(node.left, results)
        results.append(node.data)
        if node.right:
            self.inorder_traversal(node.right, results)
        return results


def populate_tree_from_db(db):
    tree = BinarySearchTree()
    three_hours_ago = datetime.now(pytz.utc) - timedelta(hours=3)
    print("Three hours ago:", three_hours_ago)
    three_hours_timestamp = int(three_hours_ago.timestamp() / 1000)
    print("Three hours ago:", three_hours_timestamp)
    #fetch metadata from mongoDB

    total_docs = db.Table_virtual.count_documents({})
    print("Total documents in the collection:", total_docs)
    count = db.Table_virtual.count_documents({
        "payload.timestamp": {"$gte": three_hours_timestamp}
    })
    all_data = db.Table_virtual.find({
        "payload.timestamp": {"$gte": three_hours_timestamp}
    })
    print("Total documents in the collection:", count)
    for doc in all_data:
        try:
            # Ensure timestamp is correctly converted
            timestamp = int(doc["payload"]["timestamp"])
            doc["payload"]["timestamp"] = datetime.fromtimestamp(timestamp, pytz.utc)
            tree.insert(doc["payload"], key="timestamp")
        except (KeyError, ValueError, TypeError) as e:
            print(f"Skipping document due to error: {e}")

    return tree
```

Challenges Faced and Solutions

The project faced a number of challenges, especially in user input validation, data retrieval efficiency, and unit conversions. Users might input an invalid IP address, a port, or a query; such users may disrupt normal work. The client was created with enhanced query validation and error-handling features that allowed the user to retry the option and also clearly presented any feedback. At the server side, having big data stored in MongoDB raised concerns about the efficiency of timely responses. A Binary Search Tree in the implementation substantially optimized the in-memory fetching of data, which significantly reduced query execution time. Another obstacle was the conversion of timestamps to UTC, which required precise calculations to align with user expectations. This was resolved using the pytz library, ensuring accuracy in time zone conversions. Finally, integrating metadata from Dataniz required careful mapping of device-specific fields, which occasionally contained missing or inconsistent data. Robust error-handling routines were implemented to skip problematic records while logging errors for future review. These solutions collectively contributed to the system's reliability and efficiency.

Feedback for Dataniz.com

While the metadata provided by Dataniz.com was important to enhance system functionality, several areas of improvement were realized in the integration process. The management of metadata directly in the database was a bit inconvenient to use, especially for operations such as deleting records that were not used. A friendlier interface for metadata management would ease these operations. Another limitation was the lack of built-in unit conversion tools in Dataniz, whereby developers had to create custom solutions for converting data to imperial units. If integrated directly into the platform, development overhead would be reduced. In addition, the user interface of the platform could be further enhanced in terms of navigation and error handling. For example, step-by-step tutorials or guided workflows on how to set up metadata would reduce the learning curve for a new user. These challenges notwithstanding, Dataniz is a very useful tool, and addressing these suggestions would go a long way in enhancing its use and attractiveness for similar IoT projects.