Design Database Assignment: Jaccob Stanton

Database Choice Justification

For AgriScan, I am choosing **MongoDB** as the database technology. This document-oriented NoSQL database is an excellent choice due to the following reasons:

- **Flexible Schema**: Crop data, drone logs, and field setups can vary in structure, making a document store like MongoDB ideal for handling such flexible data formats.
- **Scalability**: MongoDB's ability to scale horizontally fits AgriScan's potential need to handle large data volumes generated from drone scans.
- **Embedded Documents**: Data structures like field scan records and crop metrics can be embedded within larger documents, optimizing data retrieval for complex queries.

Data Structures and Their Purposes

1. Field Scans Collection

This collection holds data for each drone scan conducted on a field. It records critical metrics such as crop health, soil moisture, and growth height.

Document Structure:

```
{
 "_id": "string (auto-generated ID by MongoDB)",
 "fieldId": "string (reference ID to the Fields collection)",
 "scanDate": "string (ISO 8601 formatted timestamp)",
 "droneId": "string (reference ID to the Drones collection)",
 "cropMetrics": {
  "moistureLevel": "number",
  "chlorophyllContent": "number",
  "plantHeight": "number"
 },
 "alerts": [
  {
   "alertType": "string (e.g., 'low moisture')",
   "severity": "string (e.g., 'high', 'medium')",
   "recommendation": "string (e.g., 'Increase irrigation')"
  }
 ]
}
```

Purpose, Implementation, and Interaction:

- **Purpose**: This collection stores all crop health data collected during each drone scan.
- **Implementation**: Each scan entry is created after the drone completes its data collection and sends the processed data to the backend.
- **Interaction**: Users will access this data via the web dashboard to view current and historical crop conditions and receive any alerts for actionable insights.

---

2. Fields Collection
This collection stores information about the fields that are scanned.

Document Structure:

```
{
 "_id": "string (auto-generated ID by MongoDB)",
 "fieldName": "string",
 "location": {
  "latitude": "number",
  "longitude": "number"
 },
 "boundaryCoordinates": [
  {
   "lat": "number",
   "lng": "number"
  }
 ],
 "cropType": "string (e.g., 'corn', 'wheat')",
 "userId": "string (reference ID to the Users collection)"
}
```

Purpose, Implementation, and Interaction:

- **Purpose**: This collection records all fields set up by users for monitoring.
- **Implementation**: Users define fields via the dashboard, which are then saved to this collection.
- **Interaction**: Users will interact with this collection to set up new fields, edit existing fields, and view field-specific metrics and historical data.

---

3. Drones Collection
This collection stores information about the drones used in field scans.

Document Structure:

```
{
 "_id": "string (auto-generated ID by MongoDB)",
 "droneModel": "string",
 "serialNumber": "string",
 "status": "string (e.g., 'active', 'maintenance')",
 "lastMaintenanceDate": "string (ISO 8601 formatted timestamp)"
}
```

Purpose, Implementation, and Interaction:

- **Purpose**: This collection ensures that the drone fleet used for scanning is tracked, helping users monitor drone status and maintenance.
- **Implementation**: Drone information is managed by administrators, and scan logs reference this collection for data consistency.
- **Interaction**: Users may not directly interact with this collection but will see which drone performed which scan via the scan data.

4. Users Collection

This collection holds user data and preferences.

Document Structure:

```
{
 "_id": "string (auto-generated ID by MongoDB)",
 "username": "string",
 "email": "string",
 "passwordHash": "string (hashed for security)",
 "userRole": "string (e.g., 'farmer', 'agronomist')",
 "preferences": {
  "alertThresholds": {
   "moistureLevel": "number",
   "chlorophyllContent": "number"
  }
 }
}
```

5. Alerts History Collection

This collection maintains a log of all alerts generated for users. It helps users review past issues and actions taken.

Document Structure:

```
{
 "_id": "string (auto-generated ID by MongoDB)",
 "userId": "string (reference ID to the Users collection)",
 "fieldId": "string (reference ID to the Fields collection)",
 "scanId": "string (reference ID to the Field Scans collection)",
 "alertType": "string (e.g., 'low moisture')",
 "severity": "string (e.g., 'high', 'medium', 'low')",
 "alertMessage": "string",
 "dateIssued": "string (ISO 8601 formatted timestamp)",
 "status": "string (e.g., 'resolved', 'unresolved')",
 "actionTaken": "string (optional, user-entered action)"
}
```

Purpose, Implementation, and Interaction:

- **Purpose**: Keeps a history of alerts for reference and compliance, aiding in tracking responses and outcomes.
- **Implementation**: Each alert generated during a field scan is logged with relevant details.
- **Interaction**: Users can view alert history to track past issues and ensure proper actions have been taken.

---

6. Reports Collection

This collection stores custom reports generated by users for analyzing crop health trends over time.

Document Structure:

```
{
  "_id": "string (auto-generated ID by MongoDB)",
  "userId": "string (reference ID to the Users collection)",
  "reportName": "string",
  "fieldsIncluded": ["string (reference IDs to the Fields collection)"],
  "dateRange": {
    "start": "string (ISO 8601 formatted timestamp)",
    "end": "string (ISO 8601 formatted timestamp)"
  },
  "metricsIncluded": ["string (e.g., 'moistureLevel', 'plantHeight')"],
  "createdOn": "string (ISO 8601 formatted timestamp)",
  "dataSummary": "array (summary statistics and findings based on report parameters)"
}
```

Purpose, Implementation, and Interaction:

- **Purpose**: Allows users to generate and save reports analyzing crop data over specific time periods.
- **Implementation**: Users can create reports through the dashboard by selecting fields, date ranges, and metrics.
- **Interaction**: Users access the report generation tool, input parameters, and download or review the report directly on the platform.

---

7. Maintenance Logs Collection
This collection tracks maintenance activities performed on the drones.

Document Structure:

```
{
  "_id": "string (auto-generated ID by MongoDB)",
  "droneId": "string (reference ID to the Drones collection)",
  "maintenanceDate": "string (ISO 8601 formatted timestamp)",
  "maintenanceType": "string (e.g., 'battery replacement', 'camera calibration')",
  "performedBy": "string",
  "notes": "string"
}
```

Purpose, Implementation, and Interaction:

- **Purpose**: Keeps a record of drone maintenance for operational tracking and reliability assurance.
- **Implementation**: Administrators input maintenance records, which are referenced in drone status checks.
- **Interaction**: Maintenance logs are displayed in the drone management section, providing details on recent activities and upcoming maintenance needs.

8. Weather Data Collection
This collection stores historical and current weather data to correlate with crop performance.

Document Structure:

```
{
  "_id": "string (auto-generated ID by MongoDB)",
  "fieldId": "string (reference ID to the Fields collection)",
  "date": "string (ISO 8601 formatted timestamp)",
  "temperature": "number (in Celsius)",
  "humidity": "number (percentage)",
  "rainfall": "number (in mm)",
  "windSpeed": "number (in km/h)"
}
```

Purpose, Implementation, and Interaction:

- **Purpose**: Provides context for crop metrics by linking weather conditions to field performance.
- **Implementation**: Weather data is sourced via APIs or uploaded and linked to specific fields.

- **Interaction**: Users view weather data alongside crop metrics in reports and dashboards, helping them understand environmental impacts on crop health.

---

9. User Activity Logs Collection
This collection logs user actions within the application to enhance tracking and audit capabilities.

Document Structure:

```
{
  "_id": "string (auto-generated ID by MongoDB)",
  "userId": "string (reference ID to the Users collection)",
  "action": "string (e.g., 'created report', 'updated field')",
  "timestamp": "string (ISO 8601 formatted timestamp)",
  "details": {
    "fieldId": "string (optional reference ID)",
    "scanId": "string (optional reference ID)",
    "reportId": "string (optional reference ID)"
  }
}
```

Purpose, Implementation, and Interaction:

- **Purpose**: Provides a history of user interactions for auditing and user behavior analysis.
- **Implementation**: Logs are automatically generated upon specific user actions.
- **Interaction**: Admins and users can review logs to track activity, support troubleshooting, and ensure compliance.

Example Data Usage Scenario
Scenario: A farmer logs into the AgriScan dashboard and sets up a new field named “North Field” by entering its coordinates and selecting the crop type. This creates a document in the **Fields** collection. The drone performs a scan and uploads data, which creates an entry in the **Field Scans** collection with references to the **Fields** and **Drones** collections. The farmer reviews the scan’s crop metrics and sees a “low moisture” alert with a recommended intervention.