**Architecture Design**

1. **JSON Simulator (Producer)**
   * **Purpose**: Emit JSON data every minute.
   * **Technology**: Python.
   * **Implementation Details**:
     + Generate dynamic JSON payloads based on the required schema.
     + Use a web framework like Flask or FastAPI to expose a RESTful API endpoint for emitting the JSON.
     + Alternatively, use a message broker like RabbitMQ, Kafka, or a lightweight HTTP server.
     + Emit JSON via a push or make it available for the consumer to pull.
2. **JSON Consumer**
   * **Purpose**: Consume JSON emitted by the simulator, process it, and store it.
   * **Technology**: Python.
   * **Implementation Details**:
     + Connect to the JSON Simulator's API endpoint or message broker to fetch JSON.
     + Process JSON:
       - Perform any transformations or validations required.
       - Determine file structure or database storage logic based on the JSON content.
     + Store JSON:
       - **File-based Storage**: Save JSON to a nested folder structure (e.g., based on timestamp, categories, etc.).
       - **Database Storage**: Insert JSON into an SQLite database.
3. **Storage Layer**
   * **Option 1**: File-based Storage
     + JSON files are stored in a hierarchical folder structure (e.g., year/month/day).
     + This structure allows for easy exploration.
   * **Option 2**: SQLite Database
     + Store raw JSON strings or deserialize JSON into database columns.
     + Schema Example:
       - ID (Primary Key)
       - Timestamp
       - Processed JSON (Text)
   * Use this layer to provide consistent access to historical JSON data for the UI.
4. **UI Layer**
   * **Purpose**: Display JSON data in real-time.
   * **Technology**: React.js.
   * **Implementation Details**:
     + Use a library like Axios or Fetch API to poll the backend every minute for new JSON.
     + Display JSON dynamically in a table, card, or hierarchical view format.
     + Provide options to:
       - View raw JSON.
       - Explore processed information (if applicable).
       - Search and filter historical JSON records.
   * React State Management:
     + Use Context API or libraries like Redux to manage JSON data.
5. **Communication Flow**
   * **Backend Communication**:
     + JSON Consumer communicates with the JSON Simulator via API or message broker.
   * **Frontend Communication**:
     + React frontend connects to the backend API (JSON Consumer) to fetch new JSON and display it in real-time.

**Deployment**

* **Backend (JSON Simulator & Consumer)**
  + Host as microservices on platforms like Docker, Kubernetes, or cloud services (AWS, Azure, GCP).
* **Database**
  + SQLite for development; migrate to PostgreSQL or other scalable options for production.
* **Frontend**
  + Host React app on Vercel, Netlify, or any static hosting provider.

**High-Level Workflow**

1. **JSON Simulator**:
   * Generates and emits JSON every minute.
2. **JSON Consumer**:
   * Fetches the JSON, processes it, and stores it in the file system or SQLite database.
3. **Storage Layer**:
   * Organizes the data for real-time and historical access.
4. **UI**:
   * Fetches the latest JSON and displays it in a user-friendly interface.