ChatGPT

Dashboard Development Plan for Controlled Environment Agriculture (CEA)

Overview

The existing dashboard is written in Java and serves **greenhouse/farm operators** by visualizing lighting, environment, energy and scheduling data. To support larger research and commercial facilities, the platform should evolve into a multi-modal system that hides complexity for day-to-day farming while offering deep controls and analytics for research. This document outlines **software design patterns, hardware recommendations and general techniques** for each goal without providing implementation code. Key industry references highlight formulas and best practices for energy and environmental metrics.

1. Research Mode Toggle

- **User-experience goal:** Provide a simple "Farmer" view with a few key metrics (Name, status, PPFD, DLI and energy use) while allowing researchers to expand each light card to reveal spectrum sliders, recipe details, calibration multipliers and debugging readouts.
- **Implementation pattern:** Use a **feature-flag/toggle** mechanism instead of removing code. Feature toggles allow teams to modify system behaviour at runtime without changing the code base [1]. Store the state of the `ResearchMode` toggle per user/device (for example in the database or browser local storage). At page load, read the stored flag and default to **off**.
- **UI implementation:** Add a toggle switch in the global header (common across screens). When off, collapse each card to show only the basic fields; when on, expand the card. Use CSS classes or conditional rendering to hide or display advanced controls. Avoid bundling code by gating visibility rather than performing dynamic imports.

Software suggestions

- **Java & Web frameworks:** Continue using a Java back-end (e.g., **Spring Boot** for REST endpoints) and adopt a modern front-end such as **React** or **Vaadin**. Vaadin integrates well with Java and provides server-side rendered UI components, while React + TypeScript with Material-UI or Tailwind offers more flexibility and can be integrated with the existing Java services via REST or GraphQL.
- **Feature flag libraries:** Use open-source tools like **FF4J** (Java) or **LaunchDarkly** for dynamic feature toggling. Alternatively, implement a simple toggle table in your database, loaded into memory at startup and cached.

2. Energy Usage & DLI as First-Class Metrics

- **Metric definitions:**
- **Daily Light Integral (DLI)** is the sum of photosynthetically active radiation over a day. It is calculated using the formula **DLI = PPFD × (3600 × photoperiod) ÷ 1,000,000** [2]. Use this to compute daily, 7-day and 30-day DLI for each device or group.

- **Energy (kWh)** for a device or group can be calculated by converting watts to kilowatts and multiplying by hours of operation. For example, a 100 W device running for 10 hours uses **1 kWh** (0.1 kW × 10 h) [3].
- **Data capture:**
- **Preferred:** Install DIN-rail energy meters with current-transformer (CT) clamps that provide **three-phase power measurements via Modbus TCP/RTU**. Products like Carlo Gavazzi's **EM24 energy analyzer** support Modbus TCP/IP and measure consumption on circuits up to 65 A [4]. The gateway (e.g., an industrial Raspberry Pi or Siemens reTerminal) can poll these meters and push readings via MQTT or HTTP to the dashboard.
- **Fallback:** When external metering isn't available, estimate energy using the lighting driver's rated wattage multiplied by the driver percentage. Tag these values as estimated so users know they are less precise.
- **Visualization:** Show DLI and kWh for today, the last week and month in each light/group card. Add tooltips describing the formulas. Provide a simple **cost estimate** by multiplying kWh by a user-defined price/kWh.
- **Forecasting:** Use the schedule (period, start time, ramps) to project DLI and energy remaining for the rest of the day. The formulas above make this trivial.

Hardware recommendations

- **Energy metering:** Use multi-channel meters where each module monitors a single circuit. Choose devices that support **Modbus RTU/TCP** or **MQTT** for integration with your gateway. Consider adding **branch-circuit meters** for HVAC and dehumidifiers to get a full energy picture.
- **Connectivity:** Connect meters via **RS-485** to a Modbus-RTU-to-TCP converter or directly to the reTerminal. Use **shielded twisted-pair** cabling to reduce interference.

3. Spectral Mix Visualization

- **Concept:** A spectral power distribution (SPD) describes the radiant power per unit wavelength of a light source [5]. Instead of showing raw channel percentages, compute the **weighted sum of SPD curves** for each channel to display the actual output spectrum.
- **Data model:** Store an SPD array for each channel/driver (e.g., 400–700 nm at 10 nm increments). Provide presets for typical white, warm-white, blue and red LEDs and allow uploading vendor-supplied SPDs.
- **Computation:** For each device or group, multiply each channel's SPD curve by its current percentage (0–100 %) and sum the resulting vectors. Normalize the integrated spectrum (e.g., divide by the maximum value) and update the bar display once per minute.
- **UI:** Replace the simple "% per driver" bar with a **stacked bar chart** showing the weighted SPD. Use a tooltip to explain that the display is computed from channel percentages and SPD curves. For optional far-red (FR) or UV channels, add extra zones beyond 700 nm.
- **Rendering:** Use a chart library such as **Chart.js**, **D3.js**, or **Recharts** (in React) to render interactive 400–700 nm bars. For Java-centric UI (Vaadin), integrate a JavaScript chart via Vaadin's `Chart` component.

4. Linking Schedules and Groups

- **Data linking:** Use a consistent naming scheme where schedules are stored with IDs like `group:<group-name>`. When a user edits a schedule from a group card, send a `POST` or `PATCH` to `/sched` with the group ID rather than multiple per-device calls (keeping in line with your Phase-4 contract).
- **UI:** Add a schedule badge to each group card summarizing the active cycle (e.g., "1-cycle 16/8 from 18:00" or "2-cycle 6 h/6 h"). Clicking the badge opens a schedule editor.
- **Editor guidelines:** Validate input to prevent conflicting schedules. Provide a calendar-style view for clarity.

Scheduling technology

- **Back-end:** Persist schedules in a database (e.g., PostgreSQL). Use Quartz or Spring's scheduling support to trigger events for ramping and photoperiod changes.
- **Configuration:** When adding new endpoints, extend existing JSON contracts rather than breaking them. Add optional fields for new features while maintaining backwards compatibility.

5. Two-Cycle Photoperiod Editor

- **Requirement:** The editor must manage **two lighting cycles** within a 24-hour day. The second cycle should auto-calculate start and end times based on the first cycle's start and duration.
- **UI design:** Provide fields for Cycle A start time and photonic period; show Cycle B start/end automatically. Include a button "Split 24 h evenly" that sets both periods to 12 hours or evenly divides the existing photoperiod. Provide validation hints when the total hours don't sum to 24 and a quick "Fix to 24 h" action.
- **Data model:** Represent two-cycle schedules as `{"period": "2c", "cycles": [{"start": "06:00", "duration": 6}, {"start": "18:00", "duration": 6}]}`. Store this as the single source of truth.
- **Rationale:** Simple guardrails prevent drift over time and are intuitive for operators.

6. SwitchBot Live Data Integration

- **Sensors:** SwitchBot makes battery-powered temperature and humidity sensors with optional CO_2 and VPD readings. The SwitchBot Meter's compact, durable design offers **long battery life (up to two years)** and supports dewpoint and **VPD** calculations; when connected to a SwitchBot Hub, it can send real-time alerts for changes in monitored environments. The sensors use Bluetooth for local communication.
- **Ingestion:** The **reTerminal** can act as a gateway using the SwitchBot Hub Mini or BLE library. Ingest data via local APIs or MQTT and publish aggregated metrics to the `/env` endpoint. Tag each reading with its room or zone.
- **Dashboard display:**
- Add environment **HUD tiles** for each room/zone showing current temperature, relative humidity (RH), VPD and CO_2. Include a small label "source: SwitchBot" to differentiate sensors.
- Provide a "Devices" drawer listing each sensor's battery and signal level for maintenance planning.

- **Data routing:** Standardize the `/env` JSON message structure to include temperature, humidity, VPD and CO_2 fields. Use the reTerminal as the local aggregator and forward data to Azure IoT Hub or your central database.

7. All-Data Explorer and AI Copilot

- **Unified table:** Build a paginated, column-configurable table where each row is a time-stamped data point. Use **namespaced column names** such as `light.ppfd`, `env.temp`, `energy.kWh`, `nutrients.ph`, etc. Provide quick filters (today, last 7 days, last 30 days, custom range) and summary statistics (current value, daily total, daily average, weekly total, monthly total).
- **Back-end storage:** Store high-resolution time series (per second or minute) and pre-aggregated daily data in a **time-series database** such as **InfluxDB**, **TimescaleDB**, or **Azure Data Explorer**. Use the reTerminal to forward data to the cloud where queries can run.
- **AI Copilot:** Add a button "Explain trends this week." When triggered, perform correlation analyses on aggregated data (e.g., Pearson or Spearman correlation between PPFD and temperature/humidity, or between outdoor conditions and HVAC load). Present a short narrative and charts summarizing the findings. For example, highlight that high humidity increases the energy required for cooling because air conditioners must remove moisture as part of the cooling process [6]. Use existing Python libraries such as **pandas**, **scikit-learn** or **SciPy** on the server side for correlation and simple ML analyses.

8. Environment–Energy Breakdown Table

- **Objective:** Show the relationship between environmental conditions and energy consumption. For each room/zone, display columns for **temperature, RH, VPD, CO_2**, and corresponding **HVAC kWh, dehumidifier kWh, fan percentage and light kWh**. Display mini scatter plots (spark lines) inline, plotting temperature against HVAC energy and RH against dehumidifier energy.
- **Adaptive lighting indicator:** Show a flag or icon when the system automatically reduced spectrum or PPFD due to heat or humidity. This gives operators transparency about automatic adjustments.
- **Data sources:** Use branch-circuit meters to capture HVAC and dehumidifier power usage. Combine with the energy formulas above to compute consumption. Use environment sensors (SwitchBot plus CO_2 sensors) for the other columns. Provide exported CSV/Excel downloads for offline analysis.

9. Detailed Lights Table

- **Purpose:** Consolidate all light-related information in one view. Columns should include: device ID, group, recipe name, schedule summary, PPFD target/actual, DLI today, kWh today/week/month, average driver percentage per channel (CW/WW/BL/RD), blue/red ratio and optional heat index (derived from watt density and airflow). Include the new **spectrum bar** for each row.
- **Trends and links:** Provide a link or button to open a trend chart (e.g., PPFD vs growth or biomass metrics) to support the grower's decision making. Use the same formula for DLI and kWh as described above.
- **Data update frequency:** Update real-time columns every few seconds for status, and summary columns every minute or hour for aggregates. Use websockets or server-sent events (SSE) for real-time pushes rather than polling.

10. Farm-Wide Energy Drill-Down

- **Hierarchical view:** Build an interactive tree map or sunburst chart showing **Farm → Room → System (Lights, HVAC, Dehumidifiers, Pumps, IT, Other) → Device**. Each node displays its kWh consumption over the chosen time window (today, week, month) and its percentage of the total.
- **Ranked table:** Complement the tree map with a sortable table listing systems and devices by energy usage. Include trend arrows to show whether consumption is increasing or decreasing compared to the previous period.
- **AI Copilot narrative:** Add a "What changed?" button that compares energy usage week-over-week and uses correlation to highlight reasons (e.g., increased cooling load due to a heatwave, or lower lighting energy due to reduced photoperiod). Such correlation is supported by evidence that high humidity increases the energy required for cooling and dehumidification [6].

11. Sane Defaults & Guardrails

- **Two-path rule:** For every design ticket, define a **"quick win" implementation** and a **"robust version"** along with metrics to measure success. For instance, release a minimal spectrum bar first, then iterate to incorporate FR/UV zones.
- **Controller contracts:** Keep existing device control endpoints unchanged (e.g., accept `HEX12` commands with channel mappings 00–64). Provide a popover or tooltip in the UI that shows the hex mapping for developers.
- **Feature flags:** Avoid code removal or branch surgery. Gate new features (e.g., `researchMode`, `showEnergyEstimates`, `enableCopilot`) behind runtime flags loaded from configuration or a feature flag service. Feature toggles are explicitly designed for this purpose [1].
- **Debug isolation:** Keep debugging and calibration controls accessible only in research or developer modes. Document these features and provide training to avoid accidental changes by operators.

12. Suggested Equipment & Integration Patterns

Power and Energy Monitoring

- **Devices:** Use **DIN-rail energy meters** with CT clamps for single- or three-phase circuits. Meters like the **Carlo Gavazzi EM24** provide direct connection up to 65 A and support **Modbus TCP/IP** communication [4]. Choose meters with self-powered operation to reduce wiring complexity.
- **Gateway:** A robust edge computer (e.g., **Raspberry Pi Compute Module** with industrial carrier board or **Siemens Simatic reTerminal**). Equip it with RS-485 and Ethernet for Modbus RTU/TCP bridging and run **MQTT** broker software (e.g., Mosquitto) to relay data to the cloud.
- **Networking:** Use shielded RS-485 cabling for Modbus communication and connect the gateway to your LAN via Ethernet or Wi-Fi. For remote connectivity, connect the reTerminal to **Azure IoT Hub** or a similar cloud service.

Environmental Sensing

- **SwitchBot sensors:** Battery-powered temperature and humidity monitors with optional dewpoint and VPD output, long battery life (up to two years), and real-time notifications through a hub. Deploy sensors in each room or micro-zone. The hub publishes readings to the reTerminal via Bluetooth or Wi-Fi.

- **CO_2 sensors:** Add room-grade CO_2 sensors that support MQTT or HTTP to monitor carbon dioxide levels (critical for plant growth and occupant safety). Choose sensors with ±50 ppm accuracy and calibrate regularly.
- **Airflow monitoring:** Where heat is persistent (e.g., due to high watt density), install anemometers or airflow sensors to measure fan speed and air movement.

Lighting Verification

- **Handheld sensors:** Use handheld PPFD or PAR meters for spot checks and calibration. Models with spectral sensors provide more accurate readings than photodiode-based meters.
- **Fixed sensors:** Install reference sensors in each room or bench to validate that the actual PPFD matches the target. Poll these sensors periodically and log differences.

Cloud Integration

- **Azure IoT Hub:** Use this as the central ingest for telemetry. The reTerminal acts as the edge gateway, forwarding data to the IoT Hub. Cloud functions aggregate data into **Azure Time Series Insights** or **Azure Data Explorer** for long-term storage and analytics.
- **Database schema:** Extend existing endpoints (`/env`, `/sched`) to carry new sensor readings and group schedule IDs. Introduce new read-only endpoints like `/energy` to provide summaries (today/week/month) for devices/groups/rooms. Use the same JSON style as existing APIs to maintain consistency.

Implementation Checklist (High-Level)

1. **Establish feature flags:** Implement toggles for `researchMode`, `showEnergyEstimates` and `enableCopilot`. Load flags from a configuration service or database and expose them in the UI header.
2. **Extend data contracts:**
3. `/env` → include temperature, RH, VPD and CO_2 per zone.
4. `/sched` → accept `id: "group:<name>"` for group-level schedules.
5. `/energy` → provide aggregated kWh and cost for today/week/month for each device/group/room.
6. **Develop SPD management:** Build an admin page to upload SPD presets. Store per-channel SPDs in the database. Compute weighted spectra in real time and cache results.
7. **Implement two-cycle editor:** Create a schedule editor component with auto-calculations and a “Fix to 24 h” button. Validate that total cycle duration equals 24 hours before save.
8. **Build canonical tables:**
9. All Data Explorer with time filters, summaries and export.
10. Environment ↔ Energy breakdown table showing environmental metrics and power usage side-by-side.
11. Detailed Lights table with spectrum bar, DLI, kWh and other metrics.
12. **Design AI Copilot:** Use simple statistics (Pearson/Spearman correlation) on aggregated data. Create narrative summarizers to explain relationships, such as the link between high humidity and increased cooling energy demand [6]. Provide charts alongside narratives.
13. **Deploy sensors and meters:**
14. Install DIN-rail energy meters on lighting, HVAC and dehumidifier circuits. Wire CT clamps around feeders and connect them via RS-485 to the reTerminal.

15. Deploy SwitchBot sensors and a SwitchBot Hub for each room. Integrate with the reTerminal for ingestion.
16. Install CO_2 and airflow sensors where needed.
17. **Integrate with cloud services:** Connect the reTerminal to Azure IoT for secure telemetry ingestion. Configure daily aggregation jobs and ensure data retention policies suit your analytics needs.
18. **Iterate and measure:** Start with a minimal feature set (quick win), then roll out robust versions. Collect feedback from farm operators and researchers. Use metrics (e.g., adoption of research mode, reduction in energy consumption per yield) to evaluate success.

Conclusion

The roadmap outlined above provides a holistic approach to evolving your Java-based dashboard into a **research-friendly, energy-aware and data-rich platform**. The plan emphasizes runtime feature flags [1] for safe deployments, accurate energy and DLI calculations [2] [3], physics-based spectral visualization grounded in SPD theory [5], and robust integration of environmental sensors and energy meters. Implementing these patterns will help farmers and researchers visualize complex interactions between lighting, environment and energy, leading to smarter decisions and more sustainable operations.

---

[1] Feature Toggles (aka Feature Flags)
https://martinfowler.com/articles/feature-toggles.html

[2] What is the Daily Light Integral in Crop Monitoring? - Monnit Knowledge Base
https://support.monnit.com/article/81-what-is-the-daily-light-integral-in-crop-monitoring

[3] Learn How to Calculate kWh | BKV Energy
https://bkvenergy.com/blog/how-to-calculate-kwh/

[4] EM24DINAV23XE1X
https://www.gavazziautomation.com/en-us/product/EM24DINAV23XE1X

[5] Spectral power distribution - Wikipedia
https://en.wikipedia.org/wiki/Spectral_power_distribution

[6] How Does Humidity Affect HVAC System Performance? - True Blue Heating and Cooling
https://www.trueblueac.com/blog/how-does-humidity-affect-hvac-system-performance-2/