🎨 DESIGN-AUDIT: NAS.AI WebUI Dashboard

Datum: 22.12.2025 **Reviewer:** Senior Product Designer (AI Persona) **Version:** 1.0 **Status:** 🔴 CRITICAL FINDINGS

1. 🏢 Kontext-Check

- **Produkt:** NAS.AI – High-End Network Attached Storage mit KI-Integration.
- **Zielgruppe:** Prosumer (Enthusiasten) & System-Administratoren.
- **Design-Sprache:** "Nebula 2.0" (Dark Mode, Glassmorphism, Modern).
- **Kern-Herausforderung:** Die Balance zwischen futuristischer "AI-Ästhetik" und brutalistischer "Admin-Effizienz". Ein Admin muss den Server-Status in 2 Sekunden erfassen, nicht erst schöne Farbverläufe bewundern.

2. 🚦 Design-Review (Ampel-System)

🔴 CRITICAL (UX-Killer & Semantische Fehler)

Diese Punkte gefährden die Nutzbarkeit im Ernstfall und müssen priorisiert behoben werden.

1. **Semantisches "Card-Mashing" (Security & Backup)**
   - **Fundstelle:** `Dashboard.jsx` , Card 2.
   - **Problem:** Du vermischst "Datensicherheit" (Backup/Disaster Recovery) mit "Security" (Firewall/Intrusion Detection) in einer einzigen Karte.
   - **UX-Risiko:** Das sind zwei völlig unterschiedliche mentale Modelle.
     - *Backup* = "Habe ich meine Daten noch, wenn die Platte stirbt?" (Versicherung).
     - *Security* = "Greift gerade jemand mein System an?" (Alarmanlage).
   - **Konsequenz:** Ein kritischer Security-Alert (z.B. "Failed Login Attempts: 50") geht visuell unter, wenn die Karte dominiert wird von "Nächstes Backup: Morgen".
2. **Fehlende Semantik bei Metriken (Die "Cyan-Falle")**
   - **Fundstelle:** Progress Bars für CPU/RAM.
   - **Problem:** Die Bars haben feste Farbverläufe (`from-cyan-500` / `from-violet-500`).
   - **UX-Risiko:** Eine CPU bei 10% ist Cyan (OK). Eine CPU bei 99% (System steht kurz vor dem Freeze) ist... auch Cyan.
   - **Konsequenz:** Das Dashboard lügt. Es signalisiert durch die freundlichen Farben "Alles in Ordnung", auch wenn das System brennt. Ein Admin braucht hier **Ampel-Logik** (Grün → Gelb → Rot).
3. **Kontrast-Risiko bei Glassmorphism**
   - **Fundstelle:** `GlassCard` Komponente (`bg-slate-900/40 backdrop-blur-xl`).

- **Problem:** Helle Schrift (`text-slate-400`) auf halbtransparentem Hintergrund ohne definierten "Backdrop".
- **UX-Risiko:** Je nachdem, welches Hintergrundbild der User wählt (oder wenn das Browser-Rendering spinnt), wird der Text unlesbar.
- **Konsequenz:** Verstoß gegen WCAG-Barrierefreiheitsstandards. Das Interface wirkt "schmutzig" oder schwer lesbar.

🟠 WARNING (Reibungspunkte & Ineffizienz)

Diese Punkte stören den Workflow und erhöhen die kognitive Last.

1. **Platzverschwendung durch "Welcome Header"**
   - **Fundstelle:** "Dashboard – Systemübersicht und Status".
   - **Analyse:** Ein Admin weiß, dass er auf dem Dashboard ist. Diese 150px vertikaler Platz fehlen unten für wichtige Graphen oder Logs.
   - **Lösung:** Header drastisch verkleinern oder in die Top-Navigation integrieren.
2. **Versteckte Kritische Alerts**
   - **Fundstelle:** Blueprint "Global Alert Surface" (Drawer).
   - **Analyse:** Alerts werden in einem Drawer versteckt.
   - **Kritik:** Wenn eine Festplatte ausfällt (SMART Error), darf das nicht hinter einem Klick versteckt sein. Kritische Fehler (Severity: Critical) gehören permanent sichtbar "Above the Fold" auf das Dashboard.
3. **Informationsdichte (Information Density)**
   - **Analyse:** Die aktuellen Karten nutzen sehr viel Whitespace (Padding) für sehr wenig Daten (nur ein Prozentwert und ein Balken).
   - **Kritik:** Admins bevorzugen Dichte. Zeige nicht nur "CPU: 45%", sondern idealerweise einen kleinen Sparkline-Graphen der letzten 60 Minuten direkt daneben. Der *Trend* ist wichtiger als der *Momentanwert*.

🟢 GOOD (Gut gelöst)

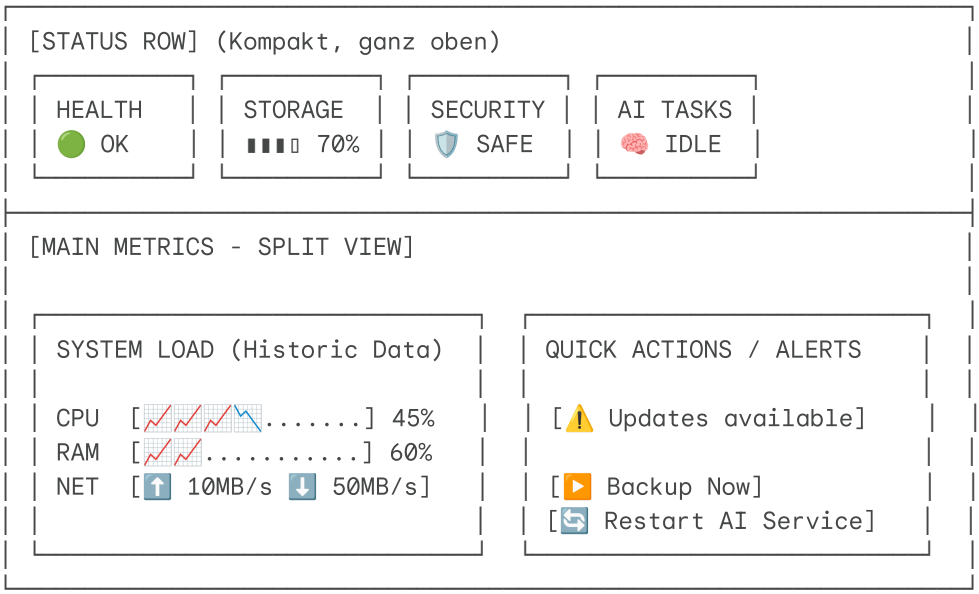
- **Skeleton Loading:** Die Nutzung von `DashboardSkeleton` verhindert Layout-Shift (CLS). Das wirkt professionell.
- **Humanized Data:** Die Funktion `getNextBackupTime` übersetzt Cron-Daten ("0 3 * * *") in Sprache ("Morgen, 3 Uhr"). Das ist exzellente UX.
- **Lokales Error Handling:** Wenn die Metriken fehlen, crasht nicht das ganze Dashboard, sondern nur die Karte zeigt einen Fehler. Vorbildlich.

3. ✨ LÖSUNGS-KONZEPT: "Nebula Admin"

Hier ist der Bauplan für die Iteration V2 des Dashboards, um die oben genannten Fehler zu beheben.

A. Layout-Struktur (The "Cockpit" Approach)

Weg von den riesigen, gleichförmigen Karten hin zu einer hierarchischen Struktur.



B. Visuelle Regeln (Design System Update)

1. **Dynamic Metric Colors (Die Ampel):** Die Progress Bars in `Dashboard.jsx` müssen eine Hilfsfunktion nutzen:
   - `< 60%` : `bg-cyan-500` (Nebula Standard / "Cool")
   - `60% - 85%` : `bg-amber-500` (Warnung / "Warm")
   - `> 85%` : `bg-rose-600` (Kritisch / "Hot") + **Pulsierender Effekt**.
2. **Hardened Glass:** Erhöhe die Opazität des Hintergrunds in `GlassCard` für besseren Kontrast:
   - Alt: `bg-slate-900/40`
   - Neu: `bg-slate-950/70` (Dunkler, weniger transparent, besser lesbar).

C. Komponenten-Trennung

Erstelle zwei dedizierte Komponenten statt der monolithischen `Dashboard.jsx` Logik:

1. `components/dashboard/SecurityStatus.jsx` : Zeigt Firewall, Failed Logins, SSL Status.
2. `components/dashboard/BackupStatus.jsx` : Zeigt *nur* Backup-Zeitpläne und Snapshot-Größen.

4. ⚖️ FAZIT

Das aktuelle Dashboard ist ein **guter visueller Prototyp ("Dribbble-ready")**, aber für den **produktiven Admin-Einsatz ("Production-ready") noch zu riskant**.

Nächste Schritte (Priorisiert):

1. [ ] **Logic Split:** Trenne Security und Backup in separate Datenströme/Karten.
2. [ ] **Safety Colors:** Implementiere die Ampel-Logik für CPU/RAM/Disk.
3. [ ] **Alert Visibility:** Bringe kritische Alerts aus dem Drawer direkt auf das Dashboard.

Ende des Reports.