### **Project "Market Pulse" - Technical Specification & Changelog (V2.0)**

This document provides a complete technical overview of the Market Pulse application and a chronological history of its development.

### **1. Project Overview & Technical Specification**

* **1.1 Objective**: A multi-tenant web application that provides hotel performance metrics via a live dashboard, allowing individual hotel clients to compare their performance against an aggregated market average.
* **1.2 Technology Stack**:
  + **Frontend**: Vanilla JavaScript (ESM), HTML5, Tailwind CSS (via CDN).
  + **Backend**: Node.js with the Express.js framework.
  + **Database**: Neon Serverless PostgreSQL.
  + **Deployment**: Vercel, with continuous deployment connected to the main branch of the GitHub repository.
  + **Authentication**: Cloudbeds OAuth 2.0 (Authorization Code Grant) for user login and onboarding. User sessions are managed by  
     express-session with a persistent connect-pg-simple store.
* **UI Interactivity (Alpine.js)**: For all new interactive pages, a **component-based approach** will be used. All UI-related state and methods will be encapsulated within a single x-data component defined in the HTML file. External .js files will be treated as pure utility modules for non-UI logic, with no direct DOM manipulation.

* **1.3 Key Application Files**:
  + server.js: The main Express.js application server, handling all API logic and serving the frontend.
  + public/app/index.html & public/dashboard.js: The primary user-facing dashboard application.
  + public/admin/index.html & public/admin.js: An administrative panel for system health checks and manual job triggers.
  + api/daily-refresh.js & api/initial-sync.js: Vercel Serverless Functions that run as background jobs to sync data from the Cloudbeds API.
* **1.4 API Endpoints Summary**:
  + **Authentication**:
    - GET /api/auth/cloudbeds: Initiates the OAuth 2.0 login flow.
    - GET /api/auth/cloudbeds/callback: Handles the OAuth redirect from Cloudbeds.
    - POST /api/admin-login: Authenticates a user for the admin panel.
  + **Dashboard Data (Session-Protected)**:
    - GET /api/kpi-summary: Provides aggregated KPI values for the dashboard cards.
    - GET /api/metrics-from-db: Fetches time-series data for the logged-in user's hotel.
    - GET /api/competitor-metrics: Fetches aggregated time-series data for the market comparison.
    - GET /api/get-hotel-name & GET /api/last-refresh-time.
  + **Admin Panel (Session-Protected)**:
    - GET /api/test-cloudbeds, GET /api/test-database, GET /api/get-all-hotels.
    - Manual Triggers: GET /api/daily-refresh, GET /api/initial-sync.
* **1.5 Database Schema Highlights**:
  + users: Stores full user profiles, including encrypted tokens and the cloudbeds\_property\_id that links a user to their specific hotel data.
  + daily\_metrics\_snapshots: Stores all time-series data, partitioned by cloudbeds\_user\_id to ensure data isolation between tenants.
  + user\_sessions: Stores persistent session data, allowing users to stay logged in.

### **2. Project Development History**

* **July 5, 2025 (Morning)**: The project was migrated from a local setup to a cloud-native solution on Vercel. The database connection was configured for production, and the background refresh script was converted into a Vercel Serverless Function scheduled via  
   vercel.json.
* **July 5, 2025 (Afternoon)**: To solve the "cold start" problem, a script was created to seed the database with mock data for five competitor hotels, establishing the "market" for comparison. The backend API was updated to aggregate this data, providing a single, averaged "market" row per day for the dashboard.
* **July 8, 2025 (V2.0 Refactor)**: A major refactoring effort was completed to convert the application to a multi-tenant platform. This involved implementing the Cloudbeds OAuth 2.0 flow, updating the database schema to be user-aware, and refactoring the API server with session-based authentication to secure all endpoints.
* **July 8, 2025 (Post-Deployment Debugging)**: After deploying the V2.0 changes, the dashboard was inaccessible due to login session failures.
  + **Diagnosis**: The root cause was identified as an issue with session persistence. The default in-memory session store was not viable in a serverless environment, causing the user's login state to be lost between API calls.
  + **Corrective Actions**: A series of fixes were implemented, including configuring CORS and cookie domains, and most critically, replacing the in-memory session store with a persistent PostgreSQL-backed store (connect-pg-simple). The database logic was also refactored to use a single, shared connection pool for stability.
  + **Final Fixes**: A subsequent TypeError on the dashboard was traced to unimplemented API endpoints (/api/metrics-from-db, /api/competitor-metrics), which were then fully implemented. Finally, the non-functional Admin Panel was restored by adding its required backend API routes.
* **Current Status (As of July 8, 2025, ~8:00 PM CEST)**: The application is stable and fully functional as a multi-tenant platform. All core architectural goals of the V2.0 migration have been met.

### **Entry: Tuesday, July 8, 2025 - 9:08 PM CEST**

**Objective:** Establish a safe and isolated local development environment to prepare for the multi-property and competitive set refactor.

**Summary of Actions:**

* **Code & Data Isolation:**
  + Created a new Git branch named feature/multi-property-and-comps to isolate all upcoming code changes from the stable main branch.
  + Created a new Neon database branch named dev-multi-property to serve as a complete, isolated checkpoint of the database schema and data, protecting the production database from any development changes.
* **Local Development Environment Configuration (server.js):**
  + **Development Login Endpoint:** Added a new POST /api/dev-login route. This endpoint is wrapped in a !isProduction check, ensuring it only exists in the local environment. It allows developers to create an authenticated session by sending a userId, bypassing the need for the live Cloudbeds OAuth flow which is tied to the production URI.
  + **CORS Policy Update:** Modified the corsOptions to conditionally add "http://localhost:3000" to the allowedOrigins array when the application is not running in a production environment. This permits API requests from the local frontend to the local server.
  + **Session Cookie Policy Update:** Modified the express-session cookie configuration to be environment-aware. It now sets sameSite: "lax" for local development (allowing cookies over HTTP) and sameSite: "none" for production (required for cross-domain OAuth).

**Current Status:** The local development environment is fully configured and operational. We are now ready to begin the database schema modifications for the multi-property and competitive set features on the isolated development branches.

**Entry: Tuesday, July 8, 2025 - 10:44 PM CEST**

**Objective:** Refactor the application to support multi-property user accounts and implement an intelligent competitive set based on a manually assigned hotel quality tier.

**Summary of Changes:**

* **Database Schema Rework:**
  + Added a  
     star\_rating column to the hotels table to store a manually assigned quality tier for each property.
  + Removed the cloudbeds\_property\_id column from the users table to de-couple users from a single property.
  + Created a new user\_properties linking table to establish a many-to-many relationship between users and their properties.
* **Backend API Refactor (server.js):**
  + The /api/auth/cloudbeds/callback endpoint was updated to query all of a user's properties from the Cloudbeds API and create a link for each one in the user\_properties table.
  + All data-fetching API endpoints (e.g., /api/kpi-summary, /api/metrics-from-db) were refactored to accept a propertyId as a query parameter and include a security check to verify the logged-in user has access to the requested property.
  + The /api/competitor-metrics endpoint was significantly enhanced. It now first determines the star\_rating of the user's selected property and then filters the market data to only include competitor hotels with a matching rating, providing a true "like-for-like" comparison.
  + A new /api/my-properties endpoint was created to provide the frontend with a list of properties a user has connected.
* **Frontend UI Development (dashboard.js & index.html):**
  + The property switcher dropdown in the dashboard header was made dynamic. It now populates by fetching data from the new /api/my-properties endpoint.
  + All data-loading functions were updated to pass the propertyId of the currently selected property from the new dropdown to the backend API, ensuring the correct data is displayed.
* **User Experience & Authentication Flow:**
  + A new /login page was created to serve as a dedicated entry point for new and returning users.
  + The authentication flow was corrected by implementing Vercel Middleware (middleware.js) to protect page routes. Logged-out users attempting to access /app/ or /admin/ are meant to be redirected to the login page - this is not working for unknown reason. Additionally, in the future logged out users shouldn’t be redirected to the oAuth stage again - this should happen only once with Cloudbeds when connecting the app for the first time. Instead after this is done we should have our own login page - either with email and password or magic link.  
      
    Also on the <https://www.market-pulse.io/admin/> page - Run Job Daily Forecast Refresh doesn’t work, resulting in a 404 error in the console.

Of course. Here is a new changelog entry summarizing the issues and the successful recovery process. You can add this to your document.

**Entry: Wednesday, July 9, 2025 - 9:30 AM CEST**

**Objective:** Diagnose and resolve a critical production failure that rendered the entire application non-functional following a branch revert.

**Summary of Actions:**

1. **Initial Diagnosis & Triage:**
   * Following a revert to the main branch, the application became unstable. The primary symptom was a total failure of the Cloudbeds OAuth login process, which crashed with a 500: FUNCTION\_INVOCATION\_FAILED error.
   * Initial troubleshooting steps, including verifying environment variables and forcing a redeployment of the local codebase to Vercel, were undertaken to rule out configuration or code synchronization issues. These steps did not resolve the problem.
2. **Root Cause Analysis:**
   * To get a definitive error message, enhanced debug logging was temporarily added to the GET /api/auth/cloudbeds endpoint in server.js.
   * A subsequent deployment and test revealed the true root cause for all failures: a ReferenceError: requirePageLogin is not defined was being thrown the moment the server tried to initialize.
   * This error was traced to two lines of code in server.js that were attempting to use a page-protection middleware function that did not exist. This remnant from a previous development effort was preventing the Express application from starting, causing all incoming requests to fail.
3. **Corrective Actions & Resolution:**
   * The two calls to the undefined requirePageLogin function were removed from the page-serving routes (/app/ and /admin/) in server.js.
   * Upon deploying this fix, all server crashes ceased. The Cloudbeds OAuth flow was restored to full functionality.
   * Subsequent testing confirmed that the previously reported 404 error on the "Daily Refresh" job trigger was also resolved by the code synchronization.

**Current Status:** The application is stable and fully functional. All issues stemming from the branch revert have been resolved. The main branches on both Git and Vercel now reflect the correct, working version of the application. Core functionalities, including user authentication, dashboard data rendering, and admin panel job triggers, are operating as expected.

**Entry: Wednesday, July 9, 2025 - 11:00 AM CEST**

**Objective:** Implement a secure, passwordless magic link login system to create a seamless user experience for returning users and enhance overall application security.

**Summary of Actions:**

1. **Email Service Integration:**
   * The SendGrid transactional email service was integrated into the application to handle the delivery of secure login links.
   * The project's domain (market-pulse.io) was authenticated by configuring the necessary CNAME and TXT DNS records, ensuring high email deliverability.
2. **Environment Configuration:**
   * A new SENDGRID\_API\_KEY environment variable was added to the Vercel project to securely connect to the email service.
3. **Database Schema Extension:**
   * A new magic\_login\_tokens table was created in the PostgreSQL database. This table is designed to store single-use, expiring tokens, ensuring each login link is secure and time-sensitive.
4. **Backend API Development:**
   * A POST /api/auth/login endpoint was created in server.js. This endpoint accepts a user's email, verifies if the user exists, and if so, triggers the email service to send the login link.
   * A GET /api/auth/magic-link-callback endpoint was created to handle the verification of the token from the user's email, create a persistent user session upon success, and grant access to the application.
5. **Frontend Rework:**
   * The login.html page was completely redesigned with a new UI and client-side JavaScript to support the email-based login flow.
   * The page now provides clear user feedback, either confirming that a link has been sent or instructing new users on how to connect via the Cloudbeds Marketplace.

**Final Status:** The magic link authentication system is yet to be tested including user flow, redirects etc.

### **Entry: Wednesday, July 9, 2025 - 11:30 AM CEST**

**Objective:** Diagnose and resolve a critical Vercel build issue that was causing frontend application scripts to fail.

**Summary of Actions:**

* **Initial Symptoms & Diagnosis:**
  + Despite a clean deployment, the application's frontend was non-functional. The browser console showed a persistent Uncaught ReferenceError: require is not defined on the first line of admin.js and dashboard.js.
  + Investigation confirmed that the source files on Vercel were correct and used modern import syntax, but the files being served to the browser were being incorrectly transpiled to use require().
* **Root Cause Analysis:**
  + The root cause was identified as an ambiguity in the project's configuration that confused Vercel's build system.
  + The catch-all route { "src": "/(.\*)", "dest": "server.js" } in vercel.json, combined with the increasing complexity of server.js, led the build system to misidentify frontend ES Module assets as backend CommonJS dependencies.
* **Corrective Actions & Resolution:**
  + To resolve this build conflict, frontend JavaScript files that use ES Module syntax were renamed to use the .mjs extension. This explicitly signals their module type to the Vercel build system, preventing incorrect transpilation.
  + The following files were renamed:
    - public/constants.js was renamed to public/constants.mjs.
    - public/admin/admin.js was renamed to public/admin/admin.mjs.
    - public/app/dashboard.js was renamed to public/app/dashboard.mjs.
  + References to these files were updated accordingly:
    - The <script> tag in public/admin/index.html was updated to point to admin.mjs.
    - The <script> tag in public/app/index.html was updated to point to dashboard.mjs.
    - The import statement in public/admin/admin.mjs was updated to import from ../constants.mjs.
* **Current Status:** The application is stable and fully functional. The build-time module conflict has been resolved, and all frontend assets are now being served correctly to the browser.

### **Entry: Wednesday, July 9, 2025 - 12:40 PM CEST**

**Objective:** Implement, test, and debug the complete passwordless magic link authentication flow for returning users.

**Summary of Actions & Resolutions:**

1. **Initial Implementation & Scaffolding:**
   * The core backend logic was implemented, including a POST /api/auth/login endpoint to generate and email a secure token, and a GET /api/auth/magic-link-callback endpoint to validate the token and create a user session.
   * A new magic\_login\_tokens table was added to the database to store the single-use, expiring tokens.
   * The SendGrid API was integrated for transactional email delivery.
   * A new login.html page was created to provide the user interface for the email login flow.
2. **Debugging Vercel Caching & Routing Conflicts:**
   * Initial testing was blocked by a persistent, incorrect redirect from the login page (/login) to the dashboard (/app/).
   * Using browser network tools, this was diagnosed as a stale redirect served directly from Vercel's Edge Cache (x-vercel-cache: HIT).
   * **Resolution:** To bypass the stuck cache, the login route was permanently changed from /login to /signin in server.js.
   * A secondary client-side redirect was discovered and traced to an erroneous <meta http-equiv="refresh"> tag in a legacy version of login.html, which was subsequently removed.
3. **Resolving Email Delivery & Security Issues:**
   * Testing revealed that magic links were causing an HTTPS security warning in the browser. This was traced to SendGrid's "Click Tracking" feature, which wrapped the secure login URL in an insecure tracking domain.
   * **Resolution:** Click Tracking was disabled in the SendGrid account settings, ensuring the link sent to users was the direct, secure URL to the application. This also improved email deliverability.
4. **Final Data Logic Correction:**
   * After resolving all redirect and link issues, users could log in but the dashboard failed to load their properties.
   * The root cause was identified as a data mismatch in the session logic. The magic link flow was incorrectly setting the session userId to the database's internal integer ID (e.g., 2), while all property data is correctly linked to the cloudbeds\_user\_id (e.g., '534490').
   * **Resolution:** The /api/auth/magic-link-callback function in server.js was refactored. It now uses the internal ID from the token to perform a secondary lookup, retrieves the correct cloudbeds\_user\_id, and sets *that* ID in the user's session.

**Final Status:** The magic link authentication system is now functional. Outstanding item - users landing on the app dashboard page should get redirected to login screen if not currently logged in - this isn’t working for some reason.

**Entry: Wednesday, July 9, 2025 - 7:45 PM CEST**

**Objective:** Overhaul the user authentication flow, enhance the user interface, and implement comprehensive legal and support documentation to create a production-ready, professional user experience.

**Summary of Actions & Resolutions:**

This was a major effort focused on stabilizing the application's core authentication logic and dramatically improving the user-facing interface and documentation.

**1. Critical Authentication & Routing Fixes:**

* **Unauthenticated Access Vulnerability:** Resolved a critical bug where unauthenticated users could access the /app/ dashboard.
  + **Diagnosis:** The express.static() middleware in server.js was serving the static index.html file before the requirePageLogin authentication check could run.
  + **Resolution:** The middleware order in server.js was corrected, moving express.static() to execute *after* all protected page routes, ensuring authentication is always checked first.
* **Infinite Redirect Loop:** Fixed a bug causing an ERR\_TOO\_MANY\_REDIRECTS error after a successful magic link login.
  + **Diagnosis:** Redundant redirect routes for /app and /admin were creating a conflict with Vercel's routing.
  + **Resolution:** The unnecessary redirect routes were removed from server.js, stabilizing the login flow.
* **Dashboard Loading Failure:** Fixed an issue where the dashboard would get stuck on a loading spinner.
  + **Diagnosis:** A 404 Not Found error for dashboard.mjs was discovered, caused by an unreliable relative path in index.html.
  + **Resolution:** The script path was changed to an absolute path (/app/dashboard.mjs) to ensure it loads correctly in all scenarios.

**2. User Experience & Feature Enhancements:**

* **Storefront Redesign:** The /signin page was completely redesigned from a simple form into a futuristic, minimalistic "storefront" marketing page. This new design includes a hero section, feature highlights, and an improved layout to clearly separate the login flow for new and returning users.
* **Functional Logout:** A "Log Out" button was implemented in the user dropdown menu on the dashboard. This button calls a new /api/auth/logout endpoint that securely destroys the user's session and redirects them to the sign-in page.
* **In-App Support & Legal Access:** "Support" and "Legal" options were added to the user dropdown menu within the dashboard, providing users with easy access to help and documentation without leaving the application.
* **Chart Rendering Fix:** A visual bug in the dashboard chart was fixed where bars could render below the zero-axis. The chart logic was updated to enforce a minimum value of 0, ensuring correct data visualization.

**3. Legal & Documentation Implementation:**

* **Terms of Service:** A comprehensive Terms of Service was drafted and implemented. It includes key clauses regarding the free nature of the current service, the right to introduce fees with 30 days' notice, and a clear data license granting Market Pulse the right to use anonymized and aggregated hotel data for commercial purposes.
* **Privacy Policy:** A detailed Privacy Policy was created, specifying the types of data collected, the use of Google Analytics, and the data retention policies for both user and hotel data.
* **Expanded FAQs:** The FAQ section was significantly expanded to proactively answer user questions about data security, disconnecting the app, the read-only nature of the service, and how to suggest new features or PMS integrations.

**Current Status:** The application is now stable, secure, and provides a polished, professional user experience from the initial landing page through to the dashboard. The core authentication lifecycle is complete, and all critical bugs identified during the session have been resolved.

### **Entry: Thursday, July 10, 2025 - 10:57 AM CEST**

**Objective:** Overhaul the application's authentication model to resolve a critical session conflict and implement a secure, scalable, database-driven role system. This effort also includes a complete redesign of the Admin Panel UI/UX to align with the main dashboard and improve usability.

**Summary of Actions:** A foundational refactor of the authentication and authorization system was completed. The previous dual-login system (magic link for users, separate password for admin) was identified as the root cause of a session corruption bug, where an admin login would overwrite a user's session ID.

The system was migrated to a unified login flow where authorization levels are stored in the database. This eliminates the session conflict and introduces a robust, single source of truth for user roles. The Admin Panel was also completely redesigned for a more professional and seamless user experience.

### **1. Core Architectural Change: Database-Driven Roles**

The most significant change was the move from a separate, hardcoded admin password to a role-based system managed within the database.

**Database Schema Update:** An is\_admin column was added to the users table to permanently flag administrator accounts.  
ALTER TABLE users ADD COLUMN is\_admin BOOLEAN DEFAULT FALSE;

* **Unified Authentication Flow:** The separate POST /api/admin-login endpoint and its associated client-side password form were completely removed. All users, including administrators, now authenticate through the same secure magic link flow.

### **2. Backend Refactor (server.js)**

The server logic was updated to support the new role-based architecture.

* **Dynamic Role Assignment on Login:** The magic link (/api/auth/magic-link-callback) and Cloudbeds OAuth (/api/auth/cloudbeds/callback) endpoints were modified. Upon successful login, they now query the users table for the is\_admin flag and store it in the user's session (e.g., req.session.isAdmin = true).
* **Session-Info Endpoint:** A new endpoint, GET /api/auth/session-info, was created. This allows the frontend to securely check the current user's role (isAdmin: true/false) without exposing any sensitive information.
* **Role-Specific API Middleware:** The single, generic requireApiLogin middleware was replaced with two specific functions to provide granular security:
  + requireUserApi: Protects all standard user data endpoints.
  + requireAdminApi: Protects all administrative endpoints (e.g., /api/get-all-hotels, /api/test-database), ensuring they can only be accessed by a session with the isAdmin flag.
* **Session Duration:** The session cookie's maxAge was extended from 30 to **60 days** to improve user convenience.

### **3. Frontend & UX Overhaul**

Significant improvements were made to the user interface and experience for both the dashboard and the admin panel.

**Admin Panel (public/admin/):**

* **Complete Redesign:** The admin page was rebuilt from the ground up to match the visual style (fonts, colors, layout) of the main dashboard, including the addition of a shared navigation sidebar.
* **Seamless Admin Access:** The admin.mjs script now uses the /api/auth/session-info endpoint on page load. If an admin session is detected, it displays the admin tools directly, bypassing the need for a password and removing the annoying "flash" of the old login form.
* **Non-Disruptive Notifications:** The browser alert() calls for job status updates were replaced with a clean, on-page status message that appears near the trigger buttons and fades after a few seconds.

**Dashboard (public/app/):**

* **Dynamic Navigation:** The dashboard.mjs script now calls /api/auth/session-info on page load. The "Admin Panel" link in the sidebar is now dynamically shown or hidden based on the user's admin status.
* **UI Polish:** A "shield" icon was added to the "Admin Panel" link, making its appearance consistent with other navigation items.

**Current Status:** The application is stable and the session conflict bug is resolved. The new role-based system is more secure, scalable, and provides a significantly improved and more professional user experience for both regular users and administrators.

Of course. Here is a new changelog entry summarizing the creation of the API Explorer tool.

### **Entry: Thursday, July 10, 2025 - 12:30 PM CEST**

**Objective:** Implement a dynamic, in-app API exploration tool to discover and visualize the full range of data available from the Cloudbeds "Data Insights" API.

**Summary of Actions & Resolutions:**

A new "Cloudbeds API Explorer" has been successfully integrated into the Admin Panel. This tool allows administrators to dynamically query the Cloudbeds API, providing a clear view of available datasets and their detailed structures, which will inform future feature development.

The implementation was a multi-step process that involved overcoming a server deployment issue and focusing on robust data presentation.

* **1. Admin Panel UI Enhancement (admin/index.html):**
  + A new "Cloudbeds API Explorer" section was added to the admin page.
  + This UI includes controls for fetching all datasets and for inspecting the structure of a specific dataset by its ID.
* **2. Backend Proxy Implementation (in server.js):**
  + To bypass a persistent Vercel deployment issue that was preventing new, separate API files from being routed correctly, a pragmatic approach was taken by adding the required logic directly to the main server.js application.
  + Two new, admin-protected endpoints were created:
    - GET /api/explore/datasets: Acts as a proxy to securely call the Cloudbeds GET /datasets endpoint, retrieving the list of all available datasets.
    - GET /api/explore/dataset-structure: A second proxy that calls the Cloudbeds GET /datasets/{id} endpoint to get the detailed field structure for a specific dataset ID passed in the query string.
* **3. Dynamic Frontend Rendering (admin.mjs):**
  + The API explorer buttons were wired up to call the new backend proxy endpoints.
  + To handle the complex JSON responses from the API, two new helper functions were created: renderDatasetsTable and renderFieldsTable.
  + These functions parse the raw JSON and dynamically render the data into clean, user-friendly HTML tables. For detailed datasets, the fields are automatically grouped by category (e.g., "Booking", "Finance") to improve readability.

**Current Status:** The API Explorer tool is fully functional. Administrators can now view all available Cloudbeds datasets and inspect the categorized field structure of any dataset directly from the Admin Panel.

### **Entry: Thursday, July 10, 2025 - 3:15 PM CEST**

**Objective:** Significantly expand the Admin Panel's API Explorer to cover both Insights and General API endpoints, add real data querying capabilities, and improve the tool's user interface and overall intelligence.

**Summary of Actions & Resolutions:**

The API Explorer tool has been evolved from a simple structure viewer into a comprehensive and intelligent development utility. It now supports the exploration of the majority of the application's authorized Cloudbeds API scopes and features a more organized, dynamic, and resilient user interface.

* **1. UI/UX Overhaul (admin/index.html):**
  + The API Explorer was redesigned into a cleaner, multi-step collapsible interface using <details> elements. This separates the various tools logically, preventing UI clutter as new features were added.
  + All necessary buttons for exploring the General API (Guest, Reservation, Hotel, Room, Rate, Taxes & Fees, User) were added to the interface.
* **2. Insights API - "Get Sample Data" Feature:**
  + A "Get Sample Data" feature was implemented to allow for querying real data from any Insights API dataset, not just viewing its structure.
  + A dynamic <textarea> was added, allowing the administrator to specify which data columns to retrieve.
  + The backend proxy (/api/explore/insights-data) was made "super-smart" through several iterations of debugging and enhancement. It can now:
    - Dynamically build a query based on a user-provided list of columns.
    - First fetch a dataset's schema to learn the data type of each field.
    - Intelligently request metrics (like sum, mean) only for numeric/currency columns, avoiding errors for dimensional fields.
    - Automatically add mandatory parameters like group\_rows or use the correct date filter (stay\_date vs. checkin\_date) based on the specific dataset\_id being queried.
* **3. General API - Full Exploration Implemented:**
  + The explorer was expanded to cover the General API. The following "Get Sample" tools were successfully implemented, each with a dedicated backend proxy in server.js that fetches and returns a single, real record from the live API:
    - **Get Sample Reservation** (using the getReservations endpoint).
    - **Get Hotel Info** (using the getHotelDetails endpoint).
    - **Get Sample Room** (using the getRooms endpoint).
    - **Get Sample Rate** (using the getRatePlans endpoint).
    - **Get Sample Guest** (using the getGuestList endpoint).
    - **Get Taxes & Fees** (using the getTaxesAndFees endpoint).
* **4. API Limitation Discovery:**
  + During testing, it was discovered that the getUsers endpoint consistently returns a scope/permission error, despite the application having the read:user scope. This indicates the endpoint requires an additional, undocumented permission. This finding has been noted for future reference.
* **5. Code Refactoring & Bug Fixes:**
  + The Cloudbeds authentication logic in server.js was refactored into a single, shared getCloudbedsAccessToken function to eliminate redundant code and improve maintainability.
  + A bug that made the "Run Endpoint Tests" button unresponsive was fixed by restoring its missing event listener in admin.mjs.

**Current Status:** The API Explorer is now a feature-complete and powerful internal tool. It provides a comprehensive and user-friendly interface for inspecting the structure and querying sample data from nearly all authorized aspects of the Cloudbeds API, fulfilling its design objective.

### **Entry: Thursday, July 10, 2025 - 9:32 PM CEST**

**Objective:** Complete a comprehensive UI/UX design and functional mock-up of the Advanced Reporting page, focusing on professional design, usability, and a feature-rich user interface before connecting to live data.

**Summary of Actions & Resolutions:**

An extensive design and layout overhaul of the reports.html page was completed. The focus of this session was to finalize the user experience and implement the UI for all desired reporting features using the existing mock data generator.

* **1. Iterative UI/UX Refinement:**
  + The layout of the report control panels was refined through several iterations based on user feedback.
  + The final design incorporates a dashboard-style, full-width control panel for date selection and presets, ensuring a consistent feel across the application.
  + The metric and option selection controls were organized into clean, distinct cards with icons and clear headings to improve usability.
* **2. New Feature Scaffolding:**
  + The user interface was built out to support significant new functionality. The following controls were added to the page as non-functional "stubs," ready to be implemented in the future:
    - **Exporting:** Buttons for "Print," "Export to CSV," and "Export to PDF."
    - **Market Data Comparisons:** An "Add Market Comparisons" toggle switch that conditionally enables "Display Order" radio buttons.
    - **Tax Display:** A toggle switch for "Show Tax-Inclusive Values."
* **3. Professional Report Table Redesign:**
  + The styling of the generated data table was completely overhauled to align with professional data reporting standards, focusing on clarity and print-readiness.
  + **Compact Design:** Spacing and padding were reduced to allow more data to fit clearly on the screen.
  + **Improved Typography:** All numerical data now uses a monospaced, right-aligned font for better readability and comparison.
  + **Custom Number Formatting:** A new formatting function was implemented to display numbers with thousands separators and apply specific decimal rules (e.g., 1 decimal for ADR, 0 for Total Revenue).
  + **Usability:** The table header is now "sticky," remaining visible when scrolling through long reports.

### **Entry: Thursday, July 10, 2025 - 11:30 PM CEST**

**Objective:** Enhance the Advanced Reporting tool with a comprehensive market comparison feature, allowing users to benchmark their performance against a competitive set with flexible and clear data visualizations.

**Summary of Actions & Resolutions:** The Advanced Reporting page has been significantly upgraded to integrate market data directly into custom reports. This was achieved by introducing dynamic delta calculations and two distinct display modes to give users powerful new analytical tools.

1. **Market Data Integration & Delta Calculation:**
   * Users can now select market-wide metrics (e.g., "Market ADR," "Market Occupancy") to display alongside their own hotel's data in a single report.
   * A "Delta" value is now automatically calculated (Your Hotel Value - Market Value) and displayed for any metric that has a corresponding market metric selected.
   * This delta is conditionally rendered, only appearing if both the user's and the market's metric are included in the report, ensuring a clean and relevant view.
   * Deltas are color-coded for immediate insight: positive performance gaps are shown in green, and negative gaps in red.
2. **Dynamic "Display Order" for Analysis:**
   * When market comparisons are enabled, a new "Display Order" control allows users to restructure the report table to suit their analytical needs:
     + **Group by Metric:** This default view organizes columns to keep all data for a single KPI together (e.g., Your Occupancy, Market Occupancy, Delta). This is ideal for direct, side-by-side comparisons of a specific metric.
     + **Group by Source:** This view reorganizes the table to show all selected "Your Hotel" metrics first, followed by all "Market" metrics and their corresponding deltas. This is useful for seeing a holistic view of your performance before looking at the market context.
3. **Enhanced Table Readability with Visual Grouping:**
   * To improve clarity, a subtle vertical border has been added to the report table when comparisons are active.
   * In the "Group by Metric" view, this border visually separates each complete metric group (e.g., the ADR group from the Occupancy group).
   * In the "Group by Source" view, it creates a single, clean dividing line between the "Your Hotel" data columns and the "Market" data columns.

**Current Status:** The market comparison and delta reporting features are fully implemented and operational. Users can now generate sophisticated, customized benchmark reports directly from the Advanced Reporting page.

### **Entry: Friday, July 11, 2025 - 9:22 AM CEST**

**Objective:** Overhaul the Advanced Reporting page to implement a new "Build First, Schedule Second" workflow, add robust CSV/PDF export capabilities, and significantly improve the overall user experience and UI design.

This was a major feature development and UX refinement session focused on making the reporting tool more intuitive, powerful, and professional.

### **1. Core Feature: Report Scheduler & Exporter**

A comprehensive set of features was added to allow users to build, export, and schedule custom reports.

* **Technology Used:**
  + **Frontend:** All new functionality was built with vanilla JavaScript (ESM), HTML5, and Tailwind CSS.
  + **PDF Generation:** The jsPDF library (v2.5.1) was used for creating PDF documents.
  + **PDF Tables:** The jsPDF-AutoTable plugin was used for rendering clean, professional tables within the PDF exports.
* **Key Functionality:**
  + **CSV Export:** Users can now export any generated report to a CSV file. The function generates a clean CSV with dynamic filenames (e.g., rockenue-partner-account-occupancy-report.csv) and correctly formatted headers and data rows.
  + **PDF Export:** A robust PDF export feature was implemented. It generates a polished document featuring a dynamic title, the report's date range, a subtle company logo, and a well-formatted table of the data.
  + **"Build First, Schedule Second" Workflow:** The user experience for scheduling reports was completely redesigned. Users now visually build their report on the main page first, then click "Create Scheduled Report" to open a lean modal that only asks for scheduling details (name, frequency, recipients, etc.).
  + **Scheduler Management:** A "Manage Schedules" modal was created, accessible via a dedicated button. This provides a centralized location for users to view and delete their saved report templates. A notification badge on the button keeps the user informed of how many schedules are active.

### **2. UI/UX Enhancements**

Significant effort was put into refining the user interface for a more intuitive and spacious feel.

* **Collapsible Report Options:** The entire "Report Options" panel is now collapsible, allowing users to hide the configuration controls to get a better view of the generated report table below.
* **Dynamic Report Titles:** The static "Generated Report Preview" title was replaced with a dynamic title that intelligently describes the report's content (e.g., *"Rockenue Partner Account Occupancy Report vs The Market"*).
* **Improved Layout & Spacing:**
  + Subtle divider lines and increased padding were added below section headers (Select Metrics, Formatting & Comparisons, etc.) to improve visual hierarchy.
  + The "Select Metrics" section was redesigned to display Hotel and Market metrics side-by-side on wider screens, making the layout more compact.
* **Action Button Redesign:**
  + The "Print Report" button was removed to streamline the UI.
  + The "Manage Schedules" button was restyled to match the other action buttons for a consistent look.
* **New Date Presets:** "Last Week" and "Current Week" options were added to the date range presets for more convenient filtering.

### **3. Debugging and Stability Fixes**

Several critical bugs were identified and resolved during the session to ensure the application is stable and reliable.

* **PDF Export Failure (Resolved):**
  + **Diagnosis:** The most critical issue was the PDF export failing with a "PDF library failed to load in time." error. This was traced to the jspdf-autotable.min.js library being blocked by the browser when loaded from an external CDN.
  + **Resolution:** The reliance on the CDN for the plugin was removed. The plugin's code was instead embedded directly into the main reports.js file, ensuring it is always available and eliminating the loading error. A robust loading state with a spinner and a non-blocking error message was also implemented for the export button.
* **Comparison Grouping (Resolved):** A bug preventing the "Group by Metric" and "Group by Source" options from correctly reordering the report columns was fixed. The table now groups data correctly based on the user's selection.
* **Script Errors (Resolved):**
  + An Incomplete or corrupt PNG file error was fixed by replacing a faulty Base64 SVG with a valid Base64 PNG for the PDF logo.
  + A TypeError that occurred on page load was resolved by adding a guard clause to ensure the updateScheduleCount function only runs when the necessary elements are present.

**Current Status:** The Advanced Reporting page is now feature-complete with respect to the session's objectives. Bug to be sorted out on PDF export

### **Entry: Friday, July 11, 2025 - 11:00 AM CEST**

**Objective**: Overhaul the Advanced Reporting page's PDF export functionality by migrating from an unstable client-side solution to a robust server-side architecture, and refine the report scheduling user experience.

**Summary of Actions & Resolutions**: This session addressed a critical failure in the PDF export feature and clarified the report scheduling workflow based on user feedback. The core of the effort was a complete architectural shift for how PDFs are generated.

**1. PDF Export: Server-Side Refactor**

* **Architectural Decision**: The existing client-side PDF generation, which relied on jsPDF and jsPDF-AutoTable, was abandoned due to persistent and unresolvable library version conflicts and browser-related errors. A decision was made to move all PDF generation to the Node.js backend for improved stability, quality, and control.
* **Technology Used**:
  + **Puppeteer**: The Puppeteer library was chosen to programmatically control a headless Chromium browser instance on the server. This allows for the creation of high-fidelity, pixel-perfect PDFs directly from HTML and CSS.
  + **Environment-Specific Packages**: To ensure compatibility with both local development and the production Vercel environment, two sets of packages were installed:
    - puppeteer: The full package, which includes a compatible Chromium browser for local development on macOS/Windows.
    - puppeteer-core & @sparticuz/chromium@123.0.1: A lightweight version of Puppeteer and a specially compressed version of Chromium designed for serverless environments like Vercel. A specific, stable version was pinned to resolve installation errors.
* **Backend Implementation (server.js)**:
  + A new API endpoint, POST /api/export-pdf, was created.
  + This endpoint was built with environment-aware logic. It checks process.env.VERCEL\_ENV to determine whether to launch the full puppeteer package (for local development) or the serverless-compatible @sparticuz/chromium package (for production).
  + The endpoint receives an HTML string from the client, launches the appropriate headless browser, sets the page content, waits for the content to be fully rendered, and then uses page.pdf() to generate a PDF buffer.
* **Frontend Implementation (reports.js)**:
  + The exportToPDF function was completely rewritten. It no longer uses any client-side PDF libraries.
  + Its new logic constructs a complete HTML document string, including all necessary CSS from the page's <style> block.
  + It sends this HTML content in the body of a fetch request to the new /api/export-pdf endpoint.
  + It handles the blob response from the server, creating a download link to allow the user to save the generated PDF file.

**2. Report Scheduling Modal: Workflow Refinement**

* **User Feedback**: Based on the feedback that scheduling reports requires selecting a *relative* date range, the "Create Scheduled Report" modal was redesigned.
* **Final Implementation**: The "Build First (Metrics), Schedule Second (Dates)" workflow was retained.
  + The modal was updated to include its own set of user-friendly, radio-button style selectors for choosing a relative date period (e.g., "Current Month," "Last Week").
  + The corresponding JavaScript in reports.js was updated to read the selected date period *from the modal* when the user saves a schedule. This ensures that recurring reports are generated for the correct relative time frame (e.g., always for the "current month" at the time of execution).

**Current Status & Known Issues**:

* The server-side architecture for PDF generation is fully implemented.
* The "Create Scheduled Report" modal is now functionally complete and correctly captures the user's intent for relative date ranges.
* **Known Bug**: There is a critical bug where the PDF file downloaded from the server is corrupt and cannot be opened. This indicates an issue in the final PDF buffer generation or transmission step within the POST /api/export-pdf endpoint that will require further debugging.

### **Entry: Friday, July 11, 2025 - 12:38 PM CEST**

**Objective:** Debug and resolve a critical bug preventing the server-side PDF export from functioning, and complete a full design and layout overhaul of the generated PDF report to meet professional standards.

**Summary of Actions & Resolutions:** This session involved a multi-stage debugging process to fix the corrupt PDF issue, followed by an iterative design phase to dramatically improve the aesthetics and readability of the final report.

**1. PDF Export - Debugging & Resolution:**

* **Initial Diagnosis & Correction:** The initial investigation focused on an empty server console log, which suggested an authentication failure. A request to inspect the browser's network activity disproved this, revealing a  
   Status Code: 200 OK. This confirmed the server was processing the request but that the file's  
   *content* was corrupt.
* **Root Cause & Final Fix:** The root cause was identified as an issue with how the binary PDF data was being transmitted by the Express server. The fix involved two key changes in server.js:
  + Replacing the high-level  
     res.send(pdfBuffer) with the more direct res.end(pdfBuffer) to ensure a clean, unmodified binary data stream.
  + Reverting a  
     page.setContent option from networkidle0 to domcontentloaded for better reliability with self-contained HTML.
* **Result:** These changes fully resolved the file corruption issue, making the PDF export feature stable and functional.

**2. PDF Design & Layout Overhaul:**

* **Modal Redesign:** The "Create Scheduled Report" modal was redesigned with a modern, two-column layout, improved visual selectors for the report period, and a larger, more user-friendly size. The "Custom Range" option was removed to streamline the scheduling workflow.
* **Report Aesthetics:** Based on iterative feedback, the design of the generated PDF document was significantly enhanced:
  + **Layout & Spacing:** The report now uses a clean, centered layout with a professional header and footer, increased padding, and a simple white background to improve readability.
  + **Logo & Branding:** The company logo is now correctly rendered in the header alongside the report title and generation date.
  + **Consistent Font Size:** A bug causing the font siz
  + **Dynamic Title Correction:** The logic for the report's title was corrected to remove the primary metric (e.g., "Rooms Sold"), resulting in a cleaner title like "Rockenue Partner Account Report vs The Market".

**Final Status:** The Advanced Reporting feature is now considered complete and production-ready. A few bugs left with the way final PDF output looks - consistent font size not working.

**Entry: Friday, July 11, 2025 - 1:46 PM CEST**

**Objective:** Overhaul the Advanced Reporting page's PDF export functionality to resolve critical layout bugs, improve usability, and create a polished, professional, and production-ready output.

**Summary of Actions & Resolutions:** A complete architectural and user experience overhaul of the PDF export feature was completed. The initial implementation suffered from several issues, including inconsistent layouts, content being cut off, and unprofessional presentation. Through an iterative process of debugging and enhancement, these issues were systematically resolved, resulting in a robust and intelligent reporting tool.

**1. Core Architectural Change: Intelligent, User-Driven Layouts** The most significant change was the implementation of a user-facing layout system that provides immediate feedback and prevents the generation of poorly formatted reports.

* **Interactive Export Menu:** The single "Export to PDF" button was replaced with a dynamic dropdown menu, offering users a clear choice between "Portrait" and "Landscape" orientations. The menu was correctly positioned to appear below the main button for a standard and intuitive user experience.
* **Client-Side Width Calculation:** A smart heuristic was implemented in reports.js to estimate the total width of the selected report columns. This calculation runs in real-time as the user selects metrics.
* **Dynamic Option Disabling:** If the calculated width exceeds a predefined threshold for a standard A4 page, the "Portrait" option in the export menu is automatically disabled and greyed out. A message appears informing the user that there are too many columns for portrait mode, guiding them to use the "Landscape" option instead.

**2. PDF Readability and Layout Fixes:** Several critical bugs affecting the final appearance of the PDF were resolved.

* **Header Abbreviation for Fit & Finish:** To solve issues with long column headers wrapping awkwardly in landscape mode, a new abbreviation system was implemented. Before the report HTML is sent to the server, the exportToPDF function in reports.js now programmatically replaces long headers with shorter, cleaner versions (e.g., "Total Revenue (You)" becomes "Revenue", "Market Occupancy" becomes "Mkt Occ %"). This ensures a compact and professional header row that always fits.
* **Non-Repeating Totals Row:** A bug causing the "Totals / Averages" row to repeat at the bottom of each page in a multi-page PDF was diagnosed and fixed. The issue was traced to the use of an HTML <tfoot> element, which PDF renderers are designed to repeat. The logic in reports.js was refactored to render the totals as a standard, styled <tr> at the end of the <tbody>, ensuring it appears only once at the very end of the report.

**3. Server-Side Enhancement for Landscape Support:** To support the new frontend options, the backend was updated accordingly.

* **Landscape Flag:** The POST /api/export-pdf endpoint in server.js was modified to accept a new landscape boolean property in the request body.
* **Conditional PDF Generation:** The server-side Puppeteer logic now reads this flag and dynamically sets the landscape option when calling page.pdf(), ensuring the server can generate the PDF in the orientation requested by the user.

**Final Status:** It’s nearly ready - still a few design bugs on exported PDFs

### **Entry: Friday, July 11, 2025 - 2:22 PM CEST**

**Objective:** Diagnose and resolve critical layout and data regressions in the server-side PDF export, and refine the reporting UI based on user feedback.

**Summary of Actions & Resolutions:**

This session was a focused effort to restore the Advanced Reporting PDF export to a previously "perfect" state and address several UI/UX issues on the reporting page.

1. **PDF Layout Restoration (Landscape Mode):**
   * **Diagnosis:** A visual comparison between a "perfect" historical PDF and the current output revealed major regressions. The current version had overly large fonts, excessive cell padding, and improperly wrapped column headers.
   * **Resolution:** The PDF generation logic in reports.js was significantly refactored. The CSS styles sent to the server were updated with smaller font sizes (7px) and reduced padding (3px 5px) to create a more compact, data-dense layout that prevents unwanted text wrapping.
2. **PDF Header and Data Correction:**
   * **Diagnosis:** The current PDF was missing critical "Delta" columns and had confusing, duplicated headers (e.g., two "ADR" columns). Furthermore, the vertical dividers between metric groups were inconsistent.
   * **Resolution:** The buildTableHeaders function in reports.js was completely rewritten. The new logic correctly reconstructs the "YOUR | MKT | DELTA" column groups. It was further refined to intelligently add a visual divider after every complete metric group (including "Rooms Sold" and "Unsold"), perfectly matching the layout of the target design.
3. **Reporting Page UI/UX Refinements:**
   * **Redundant Controls:** The main "Generate Report" button was removed from reports.html to streamline the UI, as the report already updates automatically on parameter changes.
   * **Iconography:** The icons for the "Select Metrics," "Formatting & Comparisons," and "Export & Actions" sections were replaced with more modern and intuitive SVGs to improve visual clarity.
   * **Modal Bug Fix:** A non-responsive close button on the "Create Scheduled Report" modal was fixed by adding the required JavaScript event listener in reports.js.

**Current Status:** The PDF export feature now accurately reproduces the desired professional layout with correct data grouping and visual dividers. The reporting page UI is cleaner and more intuitive.

### **Entry: Friday, July 11, 2025 - 4:15 PM CEST**

**Objective:** Overhaul the Advanced Reporting page (reports.html) to eliminate its reliance on complex, manual DOM manipulation. The goal was to refactor the page into a clean, maintainable, and fully reactive user interface where any change to a report control would instantly update the results without needing a "submit" button.

### **Summary of Actions & Resolutions**

A foundational refactoring of the Advanced Reporting page's frontend architecture was completed. We successfully migrated from an imperative JavaScript model to a modern, declarative approach using the **Alpine.js** library.

1. **Introduction of a Lightweight UI Library:**
   * **Action:** The Alpine.js library was added to reports.html via a CDN script tag.
   * **Outcome:** This provided the necessary tools for state management directly within the HTML, paving the way for the removal of complex JavaScript logic.
2. **Conversion to a Fully Reactive UI:**
   * **Action:** All interactive controls (date pickers, metric checkboxes, formatting toggles, and radio buttons) were bound to Alpine.js directives (x-data, @change, :disabled, etc.).
   * **Outcome:** This eliminated the need for dozens of getElementById calls and manual event listeners in reports.js. The page is now truly reactive; any change to any control automatically triggers the handleGenerateReport() function, providing instant feedback to the user.
3. **Complete Removal of PDF Export Functionality:**
   * **Action:** The entire server-side PDF export feature was removed from the application to reduce complexity and improve stability.
   * **Outcome:** The heavy puppeteer and @sparticuz/chromium dependencies were uninstalled from the project. The POST /api/export-pdf endpoint was deleted from server.js, and all corresponding buttons and logic were removed from reports.html and reports.js. This significantly simplified the codebase and eliminated a source of persistent bugs.
4. **Bug Fixes & Logic Correction:**
   * **"Display Order" Logic:** A bug in the buildTableHeaders function that ignored the "Group by Source" option was corrected. The table columns now reorder correctly based on user selection.
   * **Market Metrics Data:** An issue where selecting market metrics would update the table header but not show the data has been resolved. The getSelectedColumns function was rewritten to be more robust.
   * **Modal Functionality:** Broken event listeners for the "Create Schedule" and "Manage Schedules" modals were identified and correctly re-implemented, restoring full functionality.

### **Unexpected Challenges & Resolutions**

The refactoring process revealed several subtle bugs that required iterative debugging:

* **Challenge 1: JavaScript Module Scope**
  + **Problem:** The most significant unforeseen issue was that Alpine.js directives (@change) could not access the handleGenerateReport() function. This was because reports.js is loaded as a JavaScript Module (<script type="module">), which encapsulates its functions and prevents them from being globally accessible.
  + **Resolution:** The fix was to explicitly expose the necessary function to the global scope by adding window.handleGenerateReport = handleGenerateReport; to the end of reports.js. This made the function "visible" to Alpine.js in the HTML.
* **Challenge 2: Incomplete Reactivity**
  + **Problem:** Our initial attempts to make the page reactive were incomplete. Simply adding a single @change listener to a parent container did not capture events from all child elements, leading to a state where only some controls would trigger an update.
  + **Resolution:** We adopted a more direct and reliable approach by adding the @change="handleGenerateReport()" directive to every individual interactive control that needed to trigger a report refresh.

### **Summary of Overall Project Health**

* **Analysis:** The project is architecturally sound and functional, with a robust, database-driven backend and a clear separation between different application areas (dashboard, reports, admin). However, it carries significant **technical debt** in its frontend implementation and backend file structure. The primary challenge is the use of a monolithic server.js file and an imperative "vanilla JavaScript" approach for all UI interactivity, which makes the code complex and difficult to maintain as features are added.
* **Conclusion:** The successful refactoring of the reports.html page serves as a **perfect blueprint for modernizing the rest of the application**. By proving the value of a lightweight library like Alpine.js, we have established a clear and low-risk path forward for cleaning up the dashboard.mjs and admin.mjs files, which will dramatically improve the project's long-term health and maintainability.

**Entry: Friday, July 11, 2025 - 5:45 PM CEST**

**Objective:** Finalize the architectural refactor of the Advanced Reporting page, resolve critical bugs discovered in its "hybrid" state, and convert all remaining imperative logic to a single, fully reactive Alpine.js component.

**Summary of Actions & Resolutions:** Following the initial refactor, the reporting page was left in an unstable, hybrid state with several underlying bugs. This session completed the migration by diagnosing and fixing these issues and moving all remaining UI logic into a self-contained Alpine.js component, fulfilling the original objective of a truly reactive and maintainable interface.

**1. Stabilizing the Reactive Implementation:** A series of critical bugs, stemming from the incomplete initial refactor, were diagnosed and resolved:

* **Race Condition ("Two-Click" Bug):** The primary bug, which required users to click granularity buttons twice, was identified as a race condition. The fix was to refactor the handleGenerateReport() function to accept state directly as an argument, rather than reading it from a DOM that had not yet been updated by Alpine.js.
* **Unresponsive Controls:** Unresponsive date pickers and other buttons were fixed by applying the same robust pattern: updating the component's state first and passing that state directly into the handler functions.
* **Modal Scope Failure:** The "Create Schedule" and "Manage Schedules" modals were non-functional because their HTML was outside the scope of the main Alpine.js component. This was corrected by moving them inside the parent  
   div with the x-data attribute.

**2. Completing the Refactor: The Scheduler System** The final piece of imperative logic—the entire report scheduling system—was migrated to Alpine.js:

* **Centralized State:** A schedules: [] array was added to the Alpine x-data object to act as the single source of truth for all saved report schedules.
* **Dynamic List Rendering:** The static table in the "Manage Schedules" modal was replaced with a reactive <template x-for="...">. This template automatically renders the list of saved schedules and displays a "no scheduled reports" message when the array is empty. The notification badge on the trigger button is also now bound directly to the array's length.
* **Component-Based Logic:** The saveSchedule and deleteSchedule logic was moved out of reports.js and implemented as methods directly within the reportsPage() Alpine component. The form inputs in the "Create Schedule" modal were bound to state variables using x-model for two-way data binding.

**3. Final Code Cleanup:**

* All remaining getElementById calls and addEventListener listeners for the scheduler have been deleted from reports.js.
* The reports.js file is now a pure utility module. It contains only the helper functions for generating and formatting report data, with zero code for direct DOM manipulation.

**Current Status:** The Advanced Reporting page is now fully stable, and the refactor is complete. The page no longer operates in a hybrid mode; its interactivity is controlled by a single, self-contained, and declarative Alpine.js component. The codebase is now significantly cleaner and more maintainable, achieving the project's long-term health and modernization goals.

### **Entry: Saturday, July 12, 2025 - 9:03 PM CEST**

**Objective:** Modernize the main application dashboard by refactoring its frontend architecture, eliminating technical debt, and aligning it with the clean, reactive pattern established by the Advanced Reporting page.

**Summary of Actions:** Following the successful modernization of the reports page, the main dashboard has been completely overhauled. The previous implementation, which relied on imperative JavaScript to manually manipulate the DOM, has been replaced with a clean, declarative, and state-driven architecture powered by Alpine.js. This resolves the most significant source of technical debt in the application.

**1. Architectural Migration to a Reactive Model:**

* **Alpine.js Integration (index.html):** The dashboard is now initialized as a single, self-contained Alpine.js component (x-data="dashboardPage()"). All dynamic elements are bound directly to the component's state using directives like x-text, x-show, x-for, and :class.
* **Elimination of Manual DOM Manipulation:** The legacy approach of using document.getElementById and manually setting element properties has been completely removed. The UI now updates automatically and efficiently whenever a state variable changes.

**2. Code Encapsulation and Maintainability (dashboard.mjs):**

* **Self-Contained Component:** All of the dashboard's logic—including state management, API calls, data processing, and helper functions—has been consolidated into a single, clean object exported from dashboard.mjs.
* **Clean HTML Structure (index.html):** The HTML file is now purely a template. It is free of complex script blocks and is much easier to read and manage, as all of the interactive logic resides in the dedicated JavaScript module.

**3. Final Status:** The dashboard's frontend is now stable, maintainable, and significantly easier to debug or extend. This refactoring completes the modernization effort for the application's primary user-facing pages, bringing the entire frontend to a consistent, high-quality standard.

### **Entry: Saturday, July 12, 2025 - 10:38 PM CEST**

* **Objective**: The initial goal was to connect the Advanced Reporting page to live backend data. This was revised to first refactor the UI to use a single, shared header component to ensure consistency and improve maintainability before proceeding with the data connection.
* **Summary of Actions**:
  + Based on the insight that the dashboard and reports pages should not have duplicate headers, a plan was made to create a reusable header component.
  + A new directory was created for shared components: public/app/\_shared/.
  + The complete header markup from the dashboard was extracted into a new file: public/app/\_shared/header.html.
  + A new utility module, public/app/utils.mjs, was created, containing a reusable loadComponent() function designed to fetch and inject HTML components into the page.
* **Unexpected Challenges**:
  + **Critical Integration Failure**: The primary goal of integrating the new shared header into the reports.html page was unsuccessful.
  + **Diagnosis**: All attempts to modify the script in reports.html to load the new component resulted in numerous, page-breaking console errors. The most critical error was Alpine Expression Error: reportsPage is not defined.
  + **Root Cause**: The errors were traced back to a complex race condition. The Alpine.js library was attempting to initialize the UI and find the reportsPage component before our module script had successfully defined it in a way that Alpine could access. This indicates an issue with script loading order and scope.
  + **Resolution**: To maintain a stable codebase, the reports.html file has been left in its last known-good state (using mock data, without the shared header). A solution for the script initialization issue was not reached and will be the first task of the next session.
* **Current Status**:
  + The foundational files for the shared header architecture (header.html and utils.mjs) have been created but are not yet in use.
  + The reports.html page remains functional but is still using mock data and does not have the new header.
  + The core objective of connecting the reports page to live data is outstanding and is blocked by the header integration issue.
  + **Next Action**: Resolve the JavaScript race condition to successfully load the shared header into the reports.html page.

**Entry: Sunday, July 13, 2025 - 9:30 AM CEST**

**Objective:** Refactor the Advanced Reporting page (reports.html) to improve code organization and maintainability by separating all JavaScript logic from the HTML structure.

**Summary of Actions & Resolutions:** A systematic, incremental refactoring of the Advanced Reporting page's frontend was completed. The goal was to migrate all complex JavaScript implementation from a large, inline <script> block into an external reports.js file, establishing a clean separation of concerns.

* **Initial State Analysis:** The page's entire functionality, including state management, DOM manipulation, and utility functions, was defined within a single Alpine.js component inside reports.html, making the file difficult to navigate and maintain.
* **Refactoring Strategy:** An iterative "proxy" pattern was adopted. Functions were moved in logical groups from the HTML component to the reports.js file, where they were exposed as global functions on the window object. The original methods inside the HTML component were then replaced with simple one-line calls to these new global functions.
* **Phased Migration:**
  1. **Utility Functions:** Pure helper functions were moved first. This included all utilities for formatting dates, numbers, and currencies (formatDateForDisplay, formatValue), as well as the generateReportTitle and exportToCSV functions.
  2. **Stateful Functions:** More complex functions managing the component's state, specifically the report scheduler logic (saveSchedule, deleteSchedule), were migrated. To maintain functionality, these new global functions were designed to accept the Alpine component's context (this) as an argument.
  3. **Core Logic & DOM Rendering:** In the final phase, the remainder of the core logic was moved. This included the entire DOM rendering engine (renderReportTable and its helpers) and the primary event handlers (handleGenerateReport, handlePresetChange).

**Final Status:** The refactoring is complete. The reports.html file now serves as a clean template, with its <script> block containing only the necessary Alpine.js state properties and the init lifecycle hook. All complex implementation logic now resides in the reports.js module, making the codebase significantly easier to read, debug, and maintain.

**Entry: Sunday, July 13, 2025 - 11:15 AM CEST**

**Objective:** Implement a reusable, shared header component to ensure a consistent UI across the application, and integrate it into the Advanced Reporting page.

**Summary of Actions & Resolutions:** This session successfully established a component-based architecture for the application's UI. The primary focus was creating a single, shared header and resolving the technical challenges required to integrate it into existing pages.

* **1. Shared Component Architecture:**
  + A new directory,  
     public/app/\_shared/, was created to house reusable UI components.
  + Using the dashboard page as a template, a self-contained header component was created. This involved separating the component's logic (header.mjs) from its HTML structure (header.html).
  + A generic component loader utility (  
    utils.mjs) was created to handle fetching and initializing shared components on any page.
* **2. Integration & Race Condition Debugging:**
  + **Initial Failure:** The first attempt to load the shared header into the reports.html page failed with a critical pageHeader is not defined error.
  + **Diagnosis:** The issue was diagnosed as a JavaScript race condition. The loader was injecting the header's HTML, but the  
     <script> tag within that HTML was not being executed before Alpine.js tried to initialize the component.
  + **Resolution:** The architecture was corrected. The page's main script now first imports the header's logic from header.mjs and registers it with Alpine.js. Only then does it load the plain HTML, ensuring the component's logic is always available before it's needed.
* **3. Environment & Database Schema Synchronization:**
  + **Symptom:** After fixing the race condition, the header loaded but failed to display user data, and the server produced an error: column "is\_admin" does not exist.
  + **Root Cause Analysis:** By comparing the Vercel environment variables with the local .env file, we confirmed the local development server was connected to an older, stale database branch (dev-multi-property). This branch was missing the  
     is\_admin schema migration.
  + **Resolution:** The correct fix was applied by running the ALTER TABLE users ADD COLUMN is\_admin... command on the development database branch, bringing its schema in line with the application code and the production environment.
* **4. Final UI/UX Refinements:**
  + The header component's logic was updated to correctly display the user's full name by combining firstName and lastName.
  + The "Last Updated" timestamp was enhanced to include the time and a relative "time ago" calculation (e.g., (17h ago)).
  + The "Legal" link in the user menu was corrected to be a button that successfully opens the legal information modal.
  + A missing CSS rule was added to reports.html to fix the header's transparent background.

**Final Status:** The shared header component is now fully functional and correctly integrated into the Advanced Reporting page. The application's UI is more consistent and maintainable, resolving a key architectural challenge. NOTE: will need to update main dashboard index.hhml page with the new header in the future

### **Entry: Sunday, July 13, 2025 - 2:35 PM CEST**

**Objective:** Overhaul the Advanced Reporting page to replace its mock data generator with a live connection to the production database, and debug the integration to ensure all metrics are displayed accurately.

**Summary of Actions & Resolutions:** This was a multi-stage effort focused on connecting the reports.html page to the backend API, mirroring the successful data architecture of the main dashboard. The process involved updating both the backend and frontend, followed by a systematic debugging process to resolve data discrepancies and race conditions.

**1. Backend API Enhancement (server.js):**

* The API endpoints responsible for fetching report data (/api/metrics-from-db and /api/competitor-metrics) were updated.
* Their SQL queries were modified to select additional, previously-missing data columns required for full reporting, including total\_revenue, rooms\_sold, and capacity\_count.

**2. Frontend Data Integration (reports.js):**

* The mock data generator was completely removed from the frontend script.
* It was replaced with new functions (fetchYourHotelMetrics, fetchMarketMetrics) that call the updated backend API endpoints.
* The core data processing logic (processAndMergeData) was refactored to correctly handle the live data structure and to perform the necessary client-side calculation for Rooms Unsold (as capacity\_count - rooms\_sold).

**3. Iterative Debugging & Problem Solving:**

* **Race Condition:** An initial No property selected error was traced to a race condition where the report page tried to fetch data before the shared header component had finished loading the user's properties. This was resolved by implementing an event listener (property-changed) so the report would only generate after receiving a signal from the header that it was ready.
* **Data Verification:** When some metrics failed to appear without console errors, a console.log was temporarily added to inspect the raw API response. This debugging step proved that the server was not sending the new data fields.
* **Root Cause & Final Fix:** The investigation concluded that the backend changes had not been loaded by the running process. A **full restart of the local Node.js server** was performed, which successfully applied the updated SQL queries and resolved all outstanding data issues.

**Current Status:** The Advanced Reporting page is now fully functional and powered by live data from the database. It correctly fetches, calculates, and displays all selected "Your Hotel" and "Market" metrics, fulfilling the primary objective of the session.

### **Entry: Sunday, July 13, 2025 - 2:55 PM CEST**

**Objective:** Diagnose and resolve a persistent frontend bug causing the main dashboard chart to crash.

**Symptom:** After a series of rapid user interactions (e.g., changing date presets, switching between KPI cards), the main performance chart fails to render and the browser console displays a Uncaught TypeError: Cannot read properties of null (reading 'save') originating from the chart.js library.

**Analysis:** The root cause was identified as a complex **race condition** within the component's lifecycle. The error occurs when a user action (like clicking a KPI card) triggers an updateChart call at the exact moment the chart's <canvas> element has been temporarily removed from the DOM to display a loading animation. The Chart.js library's internal animation loop then attempts to operate on a non-existent canvas context, leading to the crash.

**Corrective Actions Attempted:** A series of increasingly robust fixes were implemented in the dashboard.mjs component to try and manage the chart's lifecycle more strictly:

1. **Initial Guard:** A simple isLoading.chart flag was used to prevent the updateChart function from running while data was being fetched. This proved insufficient.
2. **Proactive Destruction:** The logic was modified to explicitly call chartInstance.destroy() *before* setting the loading state, ensuring the old chart was destroyed while its canvas was still present in the DOM.
3. **DOM-Aware Guard:** A more direct check, !this.$refs.chartCanvas, was added to the updateChart function to verify the physical presence of the canvas element before attempting any operations.
4. **Render Cycle Alignment:** The chart creation logic was wrapped in requestAnimationFrame to ensure the destruction of the old chart and the creation of the new one were separated by a full browser render frame, giving the old chart's animation loop time to terminate cleanly.

**Final Status:** Despite these targeted efforts, the race condition persists, and the bug remains **unresolved**. The interaction between the Chart.js library's internal state and the reactive nature of the Alpine.js frontend is more complex than anticipated. This issue will require a more fundamental architectural review or an alternative approach to managing the chart's lifecycle.

**Entry: Sunday, July 13, 2025 - 3:48 PM CEST**

**Objective:** Resolve persistent, critical rendering bugs in the main dashboard chart by migrating from the previous library to the Apache ECharts library, and to refine the chart's final appearance.

**Summary of Actions & Resolutions:**

* **Initial Diagnosis & Rationale:** The previous charting implementation suffered from an unresolvable race condition. This resulted in frequent  
   TypeError crashes and visual animation glitches, such as a "drag" or "spill" effect when changing chart types or updating data rapidly. After multiple attempts to patch the issue failed, a decision was made to migrate to a new, more robust library to ensure stability and visual integrity.
* **1. Architectural Migration to ECharts:**
  + The previously used charting library script was removed from index.html and replaced with the Apache ECharts library.
  + The dashboard.mjs component was completely refactored. All chart logic was migrated to use the ECharts API, centered around the declarative echarts.init() and chart.setOption() methods.
* **2. Post-Migration Debugging & Refinements:**
  + **Initial Render Fix:** A bug where the chart would only appear after opening the browser's developer console was diagnosed as a timing issue. It was resolved by wrapping the initChart() call within a $nextTick() in the initializeDashboard function. This ensures the chart's container div is fully rendered and sized correctly before the library attempts to initialize.
  + **Color Palette Update:** The color scheme was updated to assign a warm yellow to the "Your Hotel" series and a clear blue to "The Market" series to match the desired design. The gradient area fill was also updated to use the new yellow color.
  + **Tooltip Formatting Fix:** The chart's tooltip was enhanced to correctly format its values. The tooltip's formatter function was modified to use the shared formatValue helper, ensuring all numbers are displayed with the appropriate currency (£) or percentage (%) symbols.

**Current Status:** The dashboard chart is now fully migrated to ECharts. It is stable, visually polished, and all previously identified rendering bugs and animation glitches have been resolved.

Of course. Here is a new changelog entry that summarizes the entire debugging process and the final architectural solution for the tooltip issue.

You can add this to your document.

### **Entry: Sunday, July 13, 2025 - 9:45 PM CEST**

**Objective:** Diagnose and resolve a fundamental architectural conflict between the ECharts and Alpine.js libraries that rendered all chart tooltip formatters non-functional.

#### **Summary of Actions & Resolutions:**

This was an extensive debugging session to resolve what initially appeared to be a simple tooltip formatting bug. The issue was traced back to a deep, silent conflict between the page's UI framework (Alpine.js) and the charting library (ECharts), requiring a significant architectural refactor to achieve a stable solution.

**1. Initial Diagnosis & Failed Attempts:**

* **Symptom:** The main dashboard chart's tooltip failed to format numbers with currency or percentage symbols. All attempts to apply a custom formatter or valueFormatter in dashboard.mjs were ignored by the chart instance, which continued to display the default, unformatted tooltip.
* **Debugging Steps:** A series of increasingly targeted fixes were attempted, all of which failed. These included:
  + Correcting potential JavaScript scope issues (this vs. self).
  + Hardcoding the tooltip formatter to return a simple string, which also had no effect.
  + Resolving a separate script-loading race condition in index.html that was causing numerous ReferenceErrors. While this stabilized the page, it did not solve the core tooltip problem.

**2. Root Cause Analysis: The Minimal Test Case**

* **The Breakthrough:** To isolate the problem, a separate test-chart.html file was created. This file initialized a basic ECharts chart with a formatted tooltip, but crucially, **without loading Alpine.js**.
* **Conclusion:** In the isolated test, the tooltip formatting worked perfectly. This definitively proved that the root cause was an architectural conflict between ECharts and the way Alpine.js manages the DOM and component lifecycle.

**3. The Architectural Solution: Decoupling**

* **Strategy:** Based on the test result, a decision was made to refactor the application to completely decouple the chart from Alpine.js's control (Option B).
* **Implementation:**
  + A new, standalone chartManager object was created within dashboard.mjs.
  + All chart-related logic, including the initChart and updateChart methods, was moved from the Alpine component into this new chartManager.
  + The Alpine component was refactored to no longer hold any chart state. It now communicates with the chart by calling the manager's methods (e.g., chartManager.update(...)) and passing in the required data as arguments.
  + The formatValue helper was converted to a standalone function so it could be shared by both the Alpine component (for the data tables) and the new chartManager.
* **Final Fix:** The refactor initially caused ReferenceError: formatValue is not defined in the HTML tables. This was resolved by adding a reference to the standalone formatValue function back onto the Alpine component's object, making it accessible to the template.

#### **Final Status:**

The architectural refactor is complete. The ECharts component is now fully isolated from Alpine.js, resolving the underlying conflict. The dashboard chart's tooltips are now correctly formatted as intended, and the application is stable.

### **Entry: Monday, July 14, 2025 - 11:45 AM CEST**

**Objective:** Implement a complete, end-to-end automated report scheduling system, allowing users to create, manage, and receive custom reports via email at set intervals.

**Summary of Actions & Resolutions:** This was a major feature implementation that involved creating new database schemas, backend APIs, a scheduled cron job, and integrating them with the existing frontend. The process included an extensive, iterative debugging phase to ensure stability and correctness.

**1. Backend & Automation Architecture:**

* **Database Schema:** A new scheduled\_reports table was created in the PostgreSQL database. It was designed to store all report configurations, including user and property IDs, selected metrics, frequency, and email recipients.
* **API Endpoints:** Three new, secure API endpoints were added to server.js to manage the lifecycle of a scheduled report:
  + POST /api/scheduled-reports: To create and save a new report schedule.
  + GET /api/scheduled-reports: To fetch a list of all reports scheduled by the logged-in user.
  + DELETE /api/scheduled-reports/:id: To delete a specific report.
* **Automated Cron Job:** A Vercel Cron Job was configured in vercel.json to trigger a new serverless function, /api/send-scheduled-reports.js, on a set schedule.
* **Dynamic Email Generation:** The /api/send-scheduled-reports.js script was built to be the core of the automation. It connects to the database to find due reports, calculates the correct date range, fetches live hotel and market data, generates a dynamic CSV file with a totals row, and emails it as an attachment using the existing SendGrid service.

**2. Frontend Integration:**

* The "Create Scheduled Report" and "Manage Schedules" modals on the reports page were connected to the new backend endpoints, making the feature fully interactive.
* The logic was updated to fetch and display a user's existing schedules upon page load, ensuring they persist between sessions.

**3. Critical Debugging and Refinement:**

The implementation process involved solving several complex issues to achieve a stable final product:

* **Database & Race Conditions:**
  + An initial race condition that prevented the report page from loading its data was resolved by reordering the initialization logic to ensure event listeners were ready before components that fire events were loaded.
  + A persistent "Unknown Hotel" bug was traced to a data type mismatch (text vs. integer) between the scheduled\_reports and hotels tables. This was definitively fixed by adding an explicit ::integer cast to the JOIN clause in the SQL query.
* **Cron Job & Routing:**
  + Initial tests failed with a 404 Not Found error. This was diagnosed as a missing route definition in vercel.json. Adding the route for /api/send-scheduled-reports.js resolved the issue.
  + A subsequent "No reports due at this time" error was traced to flawed time-matching logic in the email script, which was corrected to check for the exact hour and minute.
* **Data Formatting:**
  + A TypeError that crashed the email script was fixed by converting numeric strings from the database back into numbers using parseFloat() before formatting them for the CSV.
  + Based on final feedback, the CSV generation logic was refined to remove redundant words (e.g., "Your"), enforce a logical column order, and correctly include the "Totals / Averages" row based on the user's saved preference.

**Final Status:** The report scheduling feature is now fully functional and production-ready. All temporary test code has been removed, and the cron job is set to its final hourly schedule.

E**ntry: Monday, July 14, 2025 - 1:15 PM CEST**

**Objective:** Evolve the reporting suite into a professional-grade tool by adding multi-format export capabilities (Excel, PDF) to both the interactive reports page and the automated scheduler.

**Summary of Actions & Resolutions:** This session focused on a major feature expansion to provide users with high-quality, flexible report outputs. The foundational work was completed, including adding Excel export to the frontend and building the backend architecture to support generating multiple file types for scheduled reports.

**1. Core Feature Scaffolding: Multi-Format Reports**

* **Database Enhancement:** The scheduled\_reports table schema was successfully extended with a new attachment\_formats column (TEXT[]) to store user preferences for email attachments (CSV, XLSX, PDF).
* **Scheduler UI Update:** The "Create Scheduled Report" modal was updated with a new "Attachment Formats" section, allowing users to select one or more file types for their automated emails. The backend API was updated to correctly save these selections.
* **Client-Side Excel Export (Complete):** An "Export to Excel (.xlsx)" feature was successfully implemented on the main reports page. This uses a lightweight client-side library to provide instant, formatted Excel downloads directly in the browser.

**2. Backend Generation & Unresolved PDF Issue**

* **Backend Architecture:** The api/send-scheduled-reports.js script was significantly overhauled to support multi-format generation. The logic was updated to read the user's saved formats and conditionally call different generator functions. The CSV and Excel generation logic is in place.
* **PDF Dependencies:** The necessary dependencies for server-side PDF generation (puppeteer-core and @sparticuz/chromium) were added to the project's package.json.

**Current Status & Outstanding Issue:**

* **Working:** The interactive "Export to Excel" feature is fully functional. The scheduler can correctly generate and email **CSV and Excel** attachments.
* **Not Working (To Be Resolved):** The PDF generation for scheduled emails is **non-functional**. The scheduler cron job currently fails with a critical error: Cannot find module '@sparticuz/chromium'.
  + **Diagnosis:** This error indicates that despite the dependencies being correctly listed in package.json and package-lock.json, there is a persistent issue preventing Vercel from successfully installing them during the deployment build process. This was traced to a complex local Git repository state that prevented the package-lock.json file from being pushed correctly.
  + **Next Action:** The immediate next step for the following session will be to resolve this deployment/dependency issue to make the PDF generation functional.

### **Entry: Monday, July 14, 2025 - 3:59 PM CEST**

**Objective:** Diagnose and resolve a critical failure in the automated report scheduler's PDF generation feature.

**Summary of Actions & Resolutions:** An extensive, multi-stage investigation was conducted to fix a bug where the api/send-scheduled-reports.js cron job would crash when attempting to generate PDF attachments. After numerous attempts to resolve the underlying environmental issues failed, a strategic decision was made to remove the feature to ensure the overall stability of the application.

**1. Initial Diagnosis & Escalating Failures** The initial symptom was a cron job failure with the error: libnss3.so: cannot open shared object file. This indicated that the puppeteer browser was missing a required system library in the Vercel serverless environment.

A series of increasingly targeted fixes were attempted:

* **Dependency Updates:** The @sparticuz/chromium and puppeteer-core packages were updated to their latest versions to resolve any potential compatibility issues with the Vercel platform.
* **API Corrections:** The package update introduced several breaking API changes. The code was iteratively corrected to fix a TypeError: chromium.executablePath is not a function by changing the call to a property, and then to executablePath: await chromium.executablePath to correctly handle the promise-based API.
* **Persistent executablePath Error:** Despite the code being syntactically correct, the cron job continued to fail with An 'executablePath' or 'channel' must be specified for puppeteer-core. This proved that the chromium package was failing to download and prepare its browser binary within the Vercel environment.

**2. Final Attempts & Architectural Conclusion** To resolve the persistent failure, two final platform-level solutions were attempted:

1. The project's dependencies were migrated from @sparticuz/chromium to the lighter, more specialized **@sparticuz/chromium-min** package.
2. The vercel.json file was modified to include an includeFiles configuration, explicitly instructing the build system to package the chromium-min files with the serverless function.

When both of these standard, high-level fixes failed to resolve the error, we concluded that there was a fundamental and unresolvable incompatibility between the project's environment and the requirements of running a headless browser on Vercel.

**3. Final Decision: Feature Removal for Stability** Based on the investigation, the pragmatic decision was made to remove the PDF export feature entirely (Option A). This involved:

* Uninstalling the puppeteer-core and @sparticuz/chromium-min packages.
* Removing all PDF-related code and functions from the api/send-scheduled-reports.js script.
* Removing the "PDF" checkbox from the "Create Scheduled Report" modal in public/app/reports.html.
* Removing the associated includeFiles configuration from vercel.json.

**Final Status:** The report scheduling feature is now **stable and fully functional** for its remaining formats (CSV and Excel). The critical cron job failure has been resolved by strategically removing the unreliable component, ensuring the health and stability of the applicati

### **Entry: Monday, July 14, 2025 - 5:30 PM CEST**

**Objective**: Complete the UI standardization by refactoring the main application dashboard to use the reusable shared header component, ensuring a consistent user experience and improving long-term code maintainability.

**Summary of Actions & Resolutions**: A successful architectural refactor of the main dashboard (public/app/index.html) was completed, fully replacing its static header with the dynamic, shared component previously built for the reports page. This resolves a key piece of technical debt noted in the changelog.

**1. HTML & Script Loader Refactoring (public/app/index.html)**

* The entire hardcoded <header> element was removed from the dashboard's HTML structure and replaced with a simple placeholder <div> (<div id="header-placeholder"></div>).
* The page's module script loader was updated to import the dashboardComponent, the shared pageHeader component, and the loadComponent utility. It now correctly registers both components with Alpine.js before dynamically injecting the header's HTML, preventing any race conditions.

**2. Dashboard Logic Overhaul (public/app/dashboard.mjs)**

* **Elimination of Redundant Code**: All duplicated logic for fetching user properties, session information, and the last refresh time (the populatePropertySwitcher, checkUserRoleAndSetupNav, and fetchAndDisplayLastRefreshTime functions) was removed from the dashboard's script. This work is now exclusively handled by the shared header component.
* **Event-Driven Communication**: To decouple the components, the dashboard now listens for a property-changed custom event dispatched by the shared header whenever the user selects a new property.
* **New Handler Function**: A new handlePropertyChange() method was implemented in the dashboard component. This method is triggered by the event listener and is responsible for updating the dashboard's state and re-running all data queries to reflect the newly selected property.

**Final Status**: The refactoring is complete and successful. The main dashboard and the advanced reporting page are now architecturally aligned, sharing a single, consistent UI component for all top-level navigation and user interactions.

### **Entry: Monday, July 14, 2025 - 8:09 PM CEST**

**Objective**: Migrate the application to a new, dedicated Cloudbeds API application, resolve the resulting authentication issues, and refactor the initial data sync process to be more robust and user-friendly.

**Summary of Actions & Resolutions**: This was a multi-stage session focused on correcting the application's core connection to the Cloudbeds API and improving its data onboarding process. The work involved updating credentials, performing a deep-dive debug of the Adm in Panel, and completely overhauling the initial data sync logic.

**1. Cloudbeds App Credential Migration**

* **Action**: The application's environment variables on Vercel and in the local .env file were updated to use the new Client ID and Client Secret for the dedicated "Market Pulse" app [cite: Screenshot 2025-07-14 at 16.31.03.jpg].
* **Action**: Obsolete environment variables (ADMIN\_PASSWORD, CLOUDBEDS\_REFRESH\_TOKEN, CLOUDBEDS\_PROPERTY\_ID) were identified and removed to improve configuration hygiene.

**2. Admin Panel API Explorer - Authentication Debug & Fix**

* **Diagnosis**: After the credential update, all "API Explorer" functions in the Admin Panel began to fail with an authentication error. The root cause was identified as a flawed getCloudbedsAccessToken function in server.js that was using a stale, hardcoded global refresh token instead of the logged-in admin's session-specific token [cite: server.js].
* **Resolution**:
  + The flawed function in server.js was replaced with the correct, multi-tenant version from daily-refresh.js, which accepts a user's refreshToken as an argument.
  + All /api/explore/\* endpoints were refactored. They now correctly fetch the logged-in admin's refreshToken and a default property\_id from the database before making any calls to the Cloudbeds API, resolving the authentication conflict.

**3. Initial Data Sync - Logic Overhaul**

* **Objective**: The initial goal was to make the historical data sync more intelligent by first discovering a property's oldest record date.
* **Initial Attempt & Failure**: A new /api/first-record-date endpoint was created. However, testing revealed that the Cloudbeds Insights API does not support the limit and order parameters in the way required for this discovery, resulting in a persistent 400 Bad Request error [cite: User-provided error log].
* **Final Strategy**: The discovery approach was abandoned in favor of a simpler, more robust method. The initial-sync.js script was completely refactored to query a fixed, wide date range (15 years in the past to 1 year in the future), allowing the Cloudbeds API to simply return all data it has within that window.
* **Database Debugging**: The refactored sync process initially failed with a PostgreSQL error: operator does not exist: integer = character varying. This was traced to an incorrect JOIN condition in the SQL query. The query was corrected to join the users and user\_properties tables on the correct cloudbeds\_user\_id text field, resolving the data type mismatch.

**4. Final Status** The application is now stable and correctly configured to use its dedicated Cloudbeds API credentials. The Admin Panel's API Explorer is fully functional. The initial data sync process is now robust and capable of performing a deep historical data pull for any specified property, and the groundwork is laid for this process to be automated for new users in the future.

### **Entry: Tuesday, July 15, 2025 - 10:15 AM CEST**

**Objective**: Address significant technical debt by refactoring the monolithic server.js file into a modular, maintainable, and scalable architecture using Express Routers.

**Summary of Actions & Resolutions**: A foundational refactoring of the entire backend was completed. The primary server.js file, which had grown to include all API logic for the entire application, was successfully broken down into smaller, dedicated modules. This resolves the single largest architectural risk in the project and establishes a clean pattern for future development.

The refactoring was performed incrementally to ensure stability at each step:

**1. Creation of Shared Utilities (/api/utils/)**

* **db.js**: The PostgreSQL connection pool (pgPool) was moved into this dedicated utility to provide a single, shared database connection for the entire application.
* **middleware.js**: All custom authentication middleware functions (requireUserApi, requireAdminApi, requirePageLogin) were extracted into this file to be reused by the new routers.

**2. Modular Routers (/api/routes/)** The core API logic was separated into four distinct, feature-based router files. Each router is responsible for a specific area of the application. The router files were named with a .router.js suffix for clarity:

* **auth.router.js**: Now handles all user authentication and session management endpoints.
* **dashboard.router.js**: Contains all endpoints required by the main user dashboard and header.
* **reports.router.js**: Manages all API calls related to the scheduled reporting feature.
* **admin.router.js**: Contains all endpoints for the Admin Panel, including system health checks, manual job triggers (/daily-refresh, /initial-sync), and the entire API Explorer feature.

**3. Final server.js Structure**

* The main server.js file is now a lean setup script. Its only responsibilities are to initialize the Express server, configure core middleware (CORS, sessions), and then import and mount the new, dedicated router files. It no longer contains any direct application or API logic.

**Final Status**: The backend refactoring is complete. The new modular structure eliminates the risks of a monolithic file, improves code organization and readability, and makes the application significantly easier and safer to debug and extend.