**Note for AI Assistants: This document is the project's single source of truth. It must be maintained with the following rules: 1. All new changelog entries must be technical, detailed, and added chronologically just above the V2.0 Roadmap section. 2. The V2.0 Roadmap section must always remain at the very bottom of the document. 3. NEVER edit or remove previous entries to ensure a complete historical log. Project 'Market Pulse' - Changelog Initial document generated: Saturday, July 5, 2025 Core Project Specification Objective: A web application to provide hotel performance metrics, including a live dashboard and a "vs. The Market" comparison tool. Frontend: public/index.html and public/script.js . Vanilla JavaScript. Backend: server.js (Node.js/Express). Serves the frontend and provides a JSON API. Database: Neon PostgreSQL. Deployment: Vercel, connected to the main branch of the GitHub repository. Sensitive Data: All secrets (API keys, database URLs) are managed via a local .env file and as Environment Variables in the Vercel project settings. Entry: Saturday, July 5, 2025 - Morning Session Objective: Migrate the application from a local-only setup to a fully operational, cloud-native solution on Vercel. Key Files Modified: server.js , vercel.json , daily-refresh.js . Backend Changes: server.js : Logic updated to use process.env.DATABASE\_URL for the production database connection. daily-refresh.js : Converted to a Vercel Serverless Function format using import / export and a default handler function. Deployment & Configuration: vercel.json : Created to define build outputs and configure a cron job. Cron Job: path: "/api/daily-refresh" , schedule: "0 1 \* \* \*" to run daily-refresh.js daily at 1 AM UTC. Entry: Saturday, July 5, 2025 - Afternoon Session (Data Seeding) Objective: Solve the "cold start" problem for the "Us vs. The Market" feature by creating and importing a rich set of mock competitor data. Key Files Modified: seed-market-data.js (new file). Database Schema Changes: hotels table created to store static hotel information (name, city, coordinates). Foreign key constraint added between daily\_metrics\_snapshots.hotel\_id and hotels.hotel\_id . Backend Changes: seed-market-data.js : New Node.js script created to parse daily\_metrics\_snapshots\_5hotels.csv and perform a batch INSERT ... ON CONFLICT operation into the daily\_metrics\_snapshots table. Data Artifacts: daily\_metrics\_snapshots\_5hotels.csv : CSV file created with 3,650 rows of mock data for 5 competitor hotels. Entry: Saturday, July 5, 2025 - 2:40 PM CEST (UI & Backend Aggregation) Objective: Transform the raw competitor data into a true "like-for-like" comparison view and improve the UI for direct comparison. Key Files Modified: index.html , script.js , server.js . Backend Changes ( server.js ): /api/competitor-metrics : The SQL query in this endpoint was fundamentally changed. It now uses AVG() , SUM() , and GROUP BY stay\_date to perform data aggregation directly in the database. This ensures the API returns a single, averaged "market" row per day. Frontend Changes: index.html : Restructured to use a two-column div layout ( <div class="comparison-container"> ) to place "Your Hotel" and "Competitor Market" tables side-by-side. script.js : The renderCompetitorMetricsTable function was simplified. It no longer needs to handle multiple rows per day or a hotel\_id column. Headers were updated to "Market ADR", etc. Entry: Saturday, July 5, 2025 - 3:15 PM CEST (Granularity Feature) Objective: Implement a feature allowing users to view the comparison data with daily, weekly, or monthly granularity. Key Files Modified: index.html , script.js , server.js . Backend Changes ( server.js ): /api/metrics-from-db & /api/competitor-metrics : Both endpoints were enhanced to accept a granularity query parameter. A logic block was added to sanitize the input (e.g., 'weekly' -> 'week') and dynamically build the SQL query. For non-daily views, the SQL queries now use DATE\_TRUNC('<granularity>', stay\_date) and GROUP BY DATE\_TRUNC(...) to perform the aggregation at the requested time interval. Frontend Changes: index.html : Added a div with class toggle-group containing three buttons: [ Daily ] [ Weekly ] [ Monthly ] . Changed the Number of Days input to an End Date picker ( <input type="date" id="master-end-date"> ). script.js : Added a global variable currentGranularity to hold the state. Added setGranularity(granularity) function to handle button clicks and update the state. loadAllDbData() function updated to read from the new master-end-date input and pass the currentGranularity to the API calls. Table rendering functions updated to change the date column header based on the selected granularity. Entry: Saturday, July 5, 2025 - 3:35 PM CEST (UX & Cosmetic Refinements) Objective: Polish the user interface and improve the overall user experience. Key Files Modified: index.html , script.js . Frontend Changes: index.html : The CSS <style> block was completely overhauled with a new, more modern design system (CSS variables, refined color palette, improved typography and spacing) for a "dashboard-like" feel. The "Live API Forecast" section was moved above the "Master Controls" box. The container for property information was changed to an empty div with a button: <div id="hotel-info-container"><button onclick="fetchHotelDetails()">...</button>...</div> . script.js : The DOMContentLoaded event listener that automatically called fetchHotelDetails() was removed. The function is now only triggered by the new button's onclick event. A new block was added to the DOMContentLoaded listener to programmatically set the master-start-date to today and master-end-date to one month in the future. The DATASET\_7\_MAP constant was updated to rename the occupancy metric's name property from "Occupancy (Direct)" to "Occupancy". Entry: Saturday, July 5, 2025 - 8:27 PM CEST (Vercel Deployment Fix) Objective: Diagnose and resolve a critical deployment failure on Vercel where the application was non-functional despite working locally. Problem Summary: The deployed application on Vercel was serving a 404: NOT\_FOUND error for the root page and all static assets (e.g., script.js , dashboard.html ). This resulted in a non-interactive page where JavaScript-driven features, such as the date pickers and data loading buttons, did not work. The root cause was a vercel.json configuration that did not correctly instruct the Vercel build system on how to handle the static frontend files. Debugging & Resolution: Initial analysis of the vercel.json file suggested the routing rules were incorrectly sending all traffic to the server.js backend, which does not have access to the static files in a serverless environment. After an initial attempt to fix the routes failed, a detailed review of the Vercel Build Logs revealed the true issue: Vercel was incorrectly compiling frontend files (like script.js ) as if they were backend functions, instead of deploying them as static content. The key log line was Compiling "script.js" from ESM to CommonJS... . The final, successful fix involved adding an explicit build rule to vercel.json : { "src": "public/\*\*", "use": "@vercel/static" } . This rule forces the Vercel build system to recognize the /public directory's contents as static assets and deploy them correctly, resolving the 404 errors. Key Files Modified: vercel.json Entry: Saturday, July 5, 2025 - 10:43 PM CEST (Dashboard UI/UX Overhaul) Objective: Transition from the proof-of-concept index.html to a professional, interactive, and modern dashboard UI using dashboard.html as the new foundation. Key Files Modified: dashboard.html (renamed to index.html ), script.js (new version created as dashboard.js ), index\_old.html (backup created). Feature 1: New Visual Design A clean, light-themed design was implemented with a professional color palette and typography (Inter for UI, Manrope for data). The layout was updated to a two-panel system with a persistent sidebar. Feature 2: Interactive KPI Cards The main KPI cards (Occupancy, ADR, RevPAR) were made interactive. Clicking a KPI card now updates the charts and tables below to display data for the selected metric. The active KPI card is highlighted with a distinct visual style to provide clear user feedback. Feature 3: Advanced Charting Synchronized Tooltips: Hovering over a data point on one chart now simultaneously displays the corresponding tooltip on the other chart, allowing for direct visual comparison. Conditional Shading: The area between the "Your Hotel" and "The Market" lines on the comparison chart is now subtly shaded green when your performance is higher and red when it is lower. Dynamic Y-Axis: The Y-axis for the Occupancy chart is now fixed from 0-100% for clarity, while other metrics use an auto-scaling axis. Feature 4: Table Enhancements Synchronized Highlighting: Hovering over a row in one table now highlights the corresponding row in the other table. Dynamic Delta Column: A "Delta" column was added to the market table, which dynamically calculates and displays the difference between your hotel and the market for the currently active KPI. Bug Fixes: Fixed a bug where RevPAR was always displaying as 0 by dynamically calculating it in the backend API. Corrected a data scaling issue where the Occupancy chart line was not displaying correctly. Entry: Sunday, July 6, 2025 - Morning Session Objective: Debug a failing Vercel cron job and refactor its core logic to meet new business requirements for daily forecast refreshing. Key Files Modified: daily-refresh.js , package.json , vercel.json .**

1. **Problem Summary The Vercel cron job configured to run daily-refresh.js was failing to execute. Once triggered manually, it produced a cascade of errors, starting with module system conflicts, followed by multiple API payload errors, and finally a database write error. Furthermore, the script's fundamental purpose was misaligned with the project's goal; it was only updating yesterday's data, not the future forecast.**
2. **Debugging & Resolution Narrative The resolution process was iterative, tackling each layer of the application stack from the runtime environment down to the data processing logic. Part 1: Resolving the Module System Conflict ( ERR\_REQUIRE\_ESM ) Problem: The initial failure was an ERR\_REQUIRE\_ESM error in the Vercel logs. Diagnosis: The script was written using ES Module ( import ) syntax for the node-fetch package, but the project's Vercel environment defaults to the CommonJS module system. Solution: The node-fetch dependency in package.json was downgraded from v3+ (ESM-only) to a CommonJS-compatible version ( ^2.7.0 ). All import statements in daily-refresh.js were converted to require() . The export default statement was changed to module.exports to make the file fully CommonJS compliant. Part 2: Correcting API Payload Errors ( 400 Bad Request ) Problem: Once the script was running, the Cloudbeds API consistently rejected the request with 400 Bad Request errors. Diagnosis & Solution: By inspecting the specific error messages from the API, several issues in the request payload were fixed sequentially: Missing property\_ids : The required key was added to the payload. Incorrect Operator: The filter operator was corrected from "equal" to "equals" . Invalid Column Name: The requested column "occupancy\_direct" was corrected to "occupancy" , aligning it with the project's data map. Missing group\_rows : The mandatory key was added, which is required by the API when fetching the occupancy metric. Part 3: Fixing the Database Write Error ( Invalid Input Syntax ) Problem: After the API call succeeded, the script failed at the final step with a PostgreSQL error: invalid input syntax for type numeric: '-' . Diagnosis: The Cloudbeds API was returning a hyphen ( - ) for certain metrics that could not be calculated. The script was attempting to insert this non-numeric string into a numeric database column. Solution: A sanitizeMetric helper function was added to daily-refresh.js . This function ensures that any value received from the API is parsed, and if it's not a valid number, it defaults to 0 before being sent to the database. Part 4: Refactoring the Core Forecast Logic Problem: It was identified that the script's logic was fundamentally misaligned with the business need. It was only updating yesterday's data, while the requirement was to refresh the full 365-day future forecast daily. Solution: The script was completely refactored. The date logic was changed to calculate a rolling 365-day window from the current day forward. A new aggregateForecastData function was implemented, adapting the robust aggregation logic from initial-sync.js . This new function correctly handles multi-day bookings by summing data points for each day instead of overwriting them, fixing the final data integrity bug. The database operation was placed inside a for loop to process each of the 365 days returned by the API, performing an UPDATE for each day. Part 5: Final Timezone Configuration Problem: The cron schedule in vercel.json was not set to the desired local time (6 AM Poland). Solution: The schedule was updated to "0 4 \* \* \*" , which corresponds to 4:00 AM UTC, ensuring the job runs at the correct local time. Entry: Sunday, July 6, 2025 - 8:20 PM CEST Objective: Implement a reliable system to display the timestamp of the last successful data refresh on the dashboard. Key Files Modified: daily-refresh.js , server.js , dashboard.js . Database Schema Changes: A new system\_state table was created to act as a persistent log book. The SQL command used was: CREATE TABLE system\_state (key TEXT PRIMARY KEY, value JSONB); . This table is designed to store key-value pairs, with the last\_successful\_refresh key holding the timestamp. Backend Changes ( daily-refresh.js ): The cron job script was updated with a final step. After all forecast data has been successfully written to the database, it now executes an INSERT ... ON CONFLICT DO UPDATE query to save the current UTC timestamp into the system\_state table with the key last\_successful\_refresh . This ensures the timestamp is only ever updated on a fully successful run. Backend Changes ( server.js ): A new, lightweight API endpoint was created at /api/last-refresh-time . This endpoint connects to the database, queries the system\_state table for the last\_successful\_refresh key, and returns the stored timestamp in a JSON response. Frontend Changes ( dashboard.js ): A new async function, fetchAndDisplayLastRefreshTime , was added. This function is called when the page loads. It makes a fetch request to the new /api/last-refresh-time endpoint. On success, it formats the returned UTC timestamp into the user's local timezone (Europe/Warsaw) and displays it in the dashboard header. It includes robust try...catch error handling, which displays a fallback message with the current time if the API call fails, preventing the UI from breaking. Entry: Sunday, July 6, 2025 - 10:55 PM CEST Objective: Complete the "Version 1.0" of the dashboard by connecting all UI elements to live data, fixing all calculation and UX bugs, and implementing a professional loading experience. Key Files Modified: dashboard.html , dashboard.js , server.js . Live Data Integration: The loadDataFromAPI function in dashboard.js was fully implemented. It now fetches detailed granular data for the charts/tables and overall KPI summary data from the backend in parallel. A processAndMergeData function was created to transform the raw API data into the unified format required by the frontend rendering logic. Calculation Bug Fixes: ADR/RevPAR: A critical bug causing incorrect ADR and RevPAR calculations on weekly/monthly views (the "average of averages" problem) was fixed. The SQL queries in server.js for both /api/metrics-from-db and /api/competitor-metrics were modified to use SUM(total\_revenue) / SUM(rooms\_sold) for correct, weighted calculations. KPI Summary Accuracy: A new dedicated endpoint, /api/kpi-summary , was created to provide mathematically correct, period-wide KPIs, completely separate from the granular chart data. This ensures the KPI cards are always accurate for the entire selected date range. Occupancy Calculation: A bug where Occupancy was showing as 0% on the KPI cards was fixed. This was caused by integer division in the database. The SQL queries in server.js were updated to cast the values to NUMERIC ( ::NUMERIC ) to ensure correct floating-point division. UX & UI Polishing: KPI Card Interaction: A bug was fixed where clicking on the ADR or RevPAR cards did not update the main chart. The setActiveMetric function was corrected to re-render the chart. A subtle fade effect was also added to non-active cards to improve the visual cue for interactivity. Presets & Defaults: The date presets were updated to "Current Month", "Next Month", and "This Year". The dashboard now correctly defaults to showing the "Current Month" on page load. A timezone bug in the date formatting helper function was also fixed to prevent dates from being off by one day. Chart Visualization: A bug where the chart appeared empty when only a single day was selected was fixed. The chart now intelligently switches from a 'line' to a 'bar' type to ensure single data points are always visible. Loading Experience: A new, more elegant loading sequence was implemented. The dashboard content is now initially hidden. A clean, non-jumping, fixed-position spinner is shown on first load, after which the content smoothly fades in, preventing any "flash" of empty tables or flickering. The full-page loader is now only used for the initial page load to make subsequent interactions feel more responsive. Dynamic Titles: The dashboard now fetches the hotel's name from the database and dynamically inserts it into the main table and chart titles for a more personalized experience. Domain & Strategy: Domain: The custom domain market-pulse.io has been purchased and connected to the Vercel project. Awaiting DNS propagation for the changes to take effect globally. Next Steps: It was determined that the current single-user application is now feature-complete for its "Version 1.0". The next major development phase ("Version 2.0") will focus on preparing the application for public use and Cloudbeds certification. This will involve implementing a full OAuth 2.0 authentication flow and refactoring the application for multi-tenancy. This work will be treated as a separate, subsequent project phase. Entry: Monday, July 7, 2025 - 9:45 AM CEST Objective: Execute a major architectural refactoring of the frontend file structure to support a new, public-facing marketing website and create a clear separation of concerns between the public site, the user application, and the internal admin panel. Strategic Driver: The previous flat file structure within the /public directory was insufficient for the new strategic requirement of having a public-facing "storefront" for the market-pulse.io domain. This refactoring establishes a professional and scalable architecture that cleanly isolates the three main components of the web presence, preparing the project for future growth. New Frontend Architecture: The frontend is now organized into a three-part structure, with each component housed in its own directory to ensure independence and clarity: Marketing Site ( /public/index.html ): The new root of the website, serving as the public homepage. Web Application ( /public/app/ ): A dedicated subdirectory for the secure, behind-login user dashboard. This is the intended destination for app.market-pulse.io . Admin Panel ( /public/admin/ ): A dedicated subdirectory for the internal control panel, repurposed from the original UI. Final File Tree Structure: public/ │ ├── index.html // NEW: Marketing Homepage │ ├── app/ │ ├── index.html // User Application Dashboard │ └── dashboard.js │ ├── admin/ │ ├── index.html // Internal Admin Panel │ └── admin.js │ └── constants.js // Shared resource**

**Key File Modifications (Creations, Moves, and Renames): New File: A new public/index.html was created to serve as the marketing page. Application Move: The primary dashboard, public/dashboard.html , was moved and renamed to public/app/index.html . Its corresponding script, public/dashboard.js , was moved to public/app/dashboard.js . Admin Panel Move: The original UI file, public/index.html , was moved and renamed to public/admin/index.html . Its original script, public/script.js , was moved and renamed to public/admin/admin.js . Bug Fixes Post-Refactoring: Fixed Broken Module Imports: Problem: After moving the scripts into subdirectories, the JavaScript import for the shared constants.js file was failing with a 404 Not Found error. Root Cause: The import path ( ./constants.js ) is relative. The browser was looking for the file inside the /app and /admin subdirectories, where it does not exist. Solution: The import paths in both public/app/dashboard.js and public/admin/admin.js were corrected to ../constants.js . This ../ correctly navigates one directory level up, allowing the scripts to find the constants.js file in the root of the /public directory. Entry: Monday, July 7, 2025 - 11:20 AM CEST Objective: Overhaul the Admin Panel to serve as a comprehensive "mission control" for the application, including security, system health checks, and manual data triggers.**

1. **Security Implementation (Code-Based) Requirement: Secure the /admin path without using Vercel's paid Password Protection feature. Solution: A free, application-level password system was implemented. Frontend: admin.html was modified to display a login form by default, hiding the main content. admin.js now sends the entered password to a new backend endpoint. Backend: A new endpoint, /api/admin-login , was created in server.js . It securely compares the submitted password against a secret ADMIN\_PASSWORD stored as an environment variable in Vercel. Local Development: The local .env file was updated to include ADMIN\_PASSWORD to enable local testing.**
2. **UI/UX Unification Requirement: The admin panel and user dashboard should feel like a single, cohesive application. Solution: public/admin/index.html was completely redesigned to use the same layout shell as the main dashboard, including the persistent sidebar and header. A link to the "Admin Panel" was added to the sidebar navigation in public/app/index.html for seamless navigation.**
3. **New Admin Panel Features System Health Dashboard: A new "System Health" section was added to admin.html to provide an at-a-glance overview of the application's status. Last Refresh: Automatically checks and displays the timestamp of the last successful cron job run. Cloudbeds API Test: A "Test" button was added that calls a new /api/test-cloudbeds endpoint to verify the API connection and token validity. Database Test: A "Test" button was added that calls a new /api/test-database endpoint to verify the database connection. Manual Action Triggers: Daily Refresh: A "Run Job" button was added to manually trigger the daily forecast refresh. This calls the existing /api/daily-refresh endpoint. Initial Sync: A "Run Full Sync" button was added to trigger a complete re-sync of all historical and forecast data. This required converting initial-sync.js into a serverless function module and creating a new /api/initial-sync endpoint in server.js to run it.**
4. **Blocking Issue & Current Status Problem: After implementing the new admin endpoints, the server fails to start, crashing immediately with a TypeError . Error Log: TypeError: Missing parameter name at 2:** [**https://git.new/pathToRegexpError**](https://git.new/pathToRegexpError) **at name (/Users/karolmarcu/Documents/market-pulse/node\_modules/path-to-regexp/dist/index.js:73:19) ... Monday, July 7, 2025 - 1:20 PM CEST Entry: Major Debugging & Refactoring Session A series of critical bugs were identified and resolved, stabilizing the local development environment and significantly improving data integrity. Server Startup Crash (SOLVED) Problem: The server was failing to start, throwing a TypeError: Missing parameter name originating from the Express.js framework. Investigation: Initial debugging focused on route syntax, module conflicts in initial-sync.js , and clearing corrupted node\_modules . None of these fixed the startup crash. Root Cause: The issue was traced to an instability in the beta version of Express being used ( express@^5.1.0 ). Solution: The application was downgraded to the latest stable version of Express ( express@^4.19.2 ). This immediately resolved the startup crash. Data Integrity & Sync Logic (SOLVED) Problem: The "Full Initial Data Sync" was only populating the database with a small fraction of the historical data, starting from June 28, 2025, instead of the expected 365 days prior. Root Cause: The initial-sync.js script did not account for API pagination. It was making one request and processing only the first "page" of data returned by the Cloudbeds API. Solution: The initial-sync.js script was refactored to make API calls in a do...while loop. It now repeatedly fetches pages of data until the API confirms there are no more results, ensuring the full two-year data range is retrieved and stored. Historical Capacity Discrepancy (CLARIFIED) Observation: The dashboard showed 100% occupancy for some dates in June because the capacity\_count was 2, while the current capacity is 13. Conclusion: It was confirmed that the property's capacity was indeed 2 during that historical period. The Cloudbeds API and the sync script were functioning correctly by providing accurate point-in-time historical data. This was deemed correct behavior, not a bug. Dashboard Hotel Name (IN PROGRESS) Problem: A SyntaxError: Unexpected token '<' error occurs in the browser when the dashboard attempts to display the hotel's name. Root Cause: The frontend ( dashboard.js ) was making a fetch request to an API endpoint ( /api/get-hotel-name ) that did not exist on the backend ( server.js ). Attempted Solution: The missing /api/get-hotel-name endpoint was created and added to server.js to fetch the name from the database. The frontend was also corrected to call the proper URL. Current Status: The error persists after initial fixes. Further debugging on this specific feature has been paused. Monday, July 7, 2025 - 2:10 PM CEST Entry: Final Debugging of Dashboard Hotel Name Feature Objective: Resolve the persistent and multi-faceted bug preventing the hotel's name from being displayed on the dashboard. This completes the work noted as "IN PROGRESS" in the previous session. Problem Summary: After the fixes from the last session, the initial SyntaxError evolved into a 500 Internal Server Error , and then finally into a silent failure where no errors were thrown, but the UI still showed the default hotel name instead of the one from the database. Debugging & Resolution Narrative: The solution required a multi-step, iterative debugging process that peeled back successive layers of the problem. Part 1: Exposing the Backend Error. The 500 Internal Server Error was generic and unhelpful. The catch block in the /api/get-hotel-name endpoint in server.js was modified to include console.error(error) . This revealed the true underlying database error in the shell logs: error: column "name" does not exist . Part 2: Resolving the Column Name Confusion. The database error log directly contradicted an assumption that the column was named name . A clear screenshot of the database schema provided definitive proof that the correct column name is property\_name . This highlighted that an earlier changelog entry was incorrect. Part 3: The Final Code Audit & Fix. With the correct column name confirmed, a final audit of the code revealed one last typo. Backend ( server.js ): The code was querying the database for property\_name but was still trying to access result.rows[0].name when constructing the JSON response. This was the cause of the silent failure. The fix involved correcting the property accessor to result.rows[0].property\_name . Frontend ( dashboard.js ): The frontend was confirmed to be correctly looking for the hotelName key in the JSON object sent by the server, requiring no further changes. Final Status: The "Dashboard Hotel Name" feature is now SOLVED. Key Files Modified: server.js , dashboard.js . Monday, July 7, 2025 - 3:41 PM CEST Entry: V1.0 Polish & Finalization Objective: Complete the final user-facing features and improvements for the V1.0 application, making it stable, robust, and ready for the next phase of development.**
5. **Dashboard Polish & UX: Dynamic Market Subtitle: The hardcoded market description on the dashboard was replaced with dynamic text. The dashboard.js file now uses the competitorCount and totalCapacity data returned from the /api/competitor-metrics endpoint to generate a precise, data-driven subtitle. Improved Error Handling: The blocking alert() on data load failure was replaced with a non-blocking notification banner. A new UI element was added to index.html , and dashboard.js was updated to show this banner with a clear error message if an API call fails. Formatted Chart Tooltips: The Chart.js configuration in dashboard.js was updated with a custom callbacks function to properly format the tooltip values, adding $ for currency and % for percentages, improving data clarity. Responsive Baseline: A responsive baseline was added to public/app/index.html to ensure the dashboard is usable on mobile devices. This involved adding the viewport meta tag and using Tailwind's responsive prefixes ( lg: , md: ) to stack layout elements vertically on smaller screens.**
6. **Admin Panel Features: Hotel List Display: A new "Hotels in Database" table was added to the admin panel. This feature required a new /api/get-all-hotels endpoint in server.js to query the database and updates to admin.js to fetch and render the data, providing an at-a-glance view of all properties in the system. Key Files Modified: public/app/dashboard.js , public/app/index.html , server.js , public/admin/admin.js . Monday, July 7, 2025 - 5:41 PM CEST Entry: V2.0 Kickoff - OAuth 2.0 Implementation & Debugging Objective: Begin the V2.0 roadmap by implementing the foundational multi-user authentication system using the OAuth 2.0 Authorization Code Grant flow.**
7. **Initial Implementation: A new login page was created at /public/app/login.html to serve as the entry point for new users. A /api/auth/cloudbeds endpoint was created in server.js to redirect users to the Cloudbeds authorization URL. A /api/auth/cloudbeds/callback endpoint was created to handle the response from Cloudbeds. A users table was created in the database to store new user credentials.**
8. **Debugging the OAuth Flow: The implementation was followed by a complex debugging session to resolve a series of cascading errors. Problem 1: 404 Not Found on Authorization URL. The initial redirect failed because the URL /api/v1.1/oauth/authorize was incorrect. Resolution: The URL was corrected to /api/v1.2/oauth based on evidence from the Cloudbeds developer portal screenshots. Problem 2: invalid\_scope Error. After fixing the URL, Cloudbeds returned an invalid\_scope error. This indicated that the list of permissions being requested in the scope parameter of the URL was invalid. Resolution: An isolation test was performed by removing the scope parameter entirely. This test succeeded in retrieving an access token, proving the core authentication flow was correct and the issue was isolated to the specific scopes being requested. Problem 3: 404 Not Found on User Info Endpoint. After the token was successfully retrieved, the next step was to get user details. The initial attempt to call** [**https://hotels.cloudbeds.com/api/v1.1/me**](https://hotels.cloudbeds.com/api/v1.1/me) **failed with a 404 error. Resolution: By analyzing the official API documentation screenshots, the correct endpoint was identified as** [**https://api.cloudbeds.com/api/v1.3/userinfo**](https://api.cloudbeds.com/api/v1.3/userinfo) **.**
9. **Current Status & Next Steps: The OAuth flow is now partially working. We can successfully get an access token and use it to fetch basic user information from the /userinfo endpoint. The final step of saving the combined user and property data to the database is still pending, as the /userinfo response does not contain the property\_id . The immediate next step is to re-introduce the call to the /me endpoint to fetch the property\_id and merge the data before saving it to the users table.**

**Monday, July 7, 2025 - 7:08 PM CEST Entry: OAuth Deep Dive & Scope Resolution Objective: Perform a deep, methodical debug of the OAuth 2.0 flow to isolate and resolve the persistent invalid\_scope and subsequent data-fetching errors. Key Files Modified: server.js.**

1. **Debugging Narrative: The session involved a step-by-step, iterative process to pinpoint the exact failure point in the authentication and data retrieval sequence.**
   * **Step 1: Fixing the Callback & Routing. The initial problem was a 404 Not Found on the /api/auth/cloudbeds/callback endpoint. This was solved by implementing the endpoint logic in server.js. This led to a secondary issue where static assets like dashboard.js were not loading, which was resolved by correcting the server's catch-all routing from app.get("/\*", ...) to specific routes for /, /app, and /admin.**
   * **Step 2: Isolating the invalid\_scope Error. With routing fixed, the Cloudbeds API returned an invalid\_scope error. A methodical test was conducted:**
     + **Requesting only read:user -> Success.**
     + **Requesting read:user and read:hotel -> Success.**
     + **Adding read:data-insights-reservations -> Failure.**
     + **Adding read:data-insights-occupancy -> Failure.**
   * **Step 3: Isolating the Data Fetching Error. After confirming that read:user and read:hotel were valid scopes, a new error emerged: Failed to fetch user or property details. The Promise.all function was hiding the specific failing call. The callback logic was refactored to call the /userinfo and /me endpoints sequentially with detailed logging. This proved that both endpoints were succeeding with the limited scopes.**
2. **Final Conclusion & Current Status:**
   * **Application State: The application has been reverted to the last stable, working version. In this state, server.js requests only the read:user and read:hotel scopes. This allows users to successfully authenticate, have their details fetched from the /userinfo and /me endpoints, and be saved to the users database table. The user is then correctly redirected to a loaded dashboard.**

**Entry: Monday, July 7, 2025 - 9:55 PM CEST Objective: Finalize the investigation into the OAuth invalid\_scope error and stabilize the V2.0 authentication flow based on the findings. Key Files Modified:**

**server.js**

### **Investigation Summary & Root Cause Analysis**

**A definitive, step-by-step isolation test was conducted to identify the exact cause of the persistent**

**invalid\_scope error returned by the Cloudbeds API. The test confirmed the following:**

* **Requesting only  
   read:user -> Success.**
* **Requesting  
   read:user and read:hotel -> Success.**
* **Adding  
   read:data-insights-reservations to the scope -> Immediate Failure.**
* **Adding  
   read:data-insights-occupancy to the scope -> Immediate Failure.**

**This testing provides conclusive evidence that the V2.0 authentication failure is caused exclusively by the data-insights scopes.**

### **Final Conclusion: External Blocker Identified**

**The root cause has been identified as an external issue with the Cloudbeds platform. While the Cloudbeds developer portal shows the read:data-insights-reservations and read:data-insights-occupancy permissions as enabled for the application, their authorization server does not recognize them and rejects the request before user consent. This discrepancy is the sole blocker preventing the completion of the V2.0 multi-tenant functionality.**

### **Current Stable Status & Next Steps**

**To create a stable baseline, the application's OAuth flow has been reverted to request only the scopes that are confirmed to work:**

**read:user and read:hotel. In this state, the entire authentication and onboarding flow is successful: a user can authorize the app, their details are fetched from the**

**/userinfo and /me endpoints, a new record is created in the users table, and they are correctly redirected to the dashboard.**

**All V2.0 development is paused pending resolution of this external dependency. It might be that these scopes get only enabled for apps that pass certification, pending clarification**

**V2.0 Roadmap & Feature Ideas This section outlines the strategic direction and planned features for the next major version of the Market Pulse application. This section should always remain at the bottom of the changelog.**

1. **Core Architecture: Multi-Tenancy & Cloudbeds Certification Objective: Refactor the application from a single-user system to a multi-tenant SaaS platform capable of serving multiple, independent hotel clients and achieving official certification for the Cloudbeds Marketplace. Key Tasks: Implement OAuth 2.0: This is the highest priority. It involves replacing the current static API key system with a secure flow where new users can grant our application access to their Cloudbeds account. This will require registering Market Pulse as a developer application with Cloudbeds to receive a client\_id and client\_secret . Database Schema Refactor: The database must be updated to support multiple users. This includes creating a users table for account information and adding a user\_id foreign key to all relevant tables (e.g., hotels , daily\_metrics\_snapshots ) to ensure data is properly segregated and secured. Backend Logic Overhaul: All backend endpoints and data synchronization scripts must be updated to be user-aware, using the logged-in user's credentials to fetch and store data.**
2. **Major Feature: Advanced Reporting Module Objective: Build a dedicated "Reports" section to provide users with powerful, customizable, and exportable data views. Key Tasks (in manageable chunks): Phase 1 (Report Builder UI): Create a new page for the reporting module. Design a UI with date pickers, granularity controls, and a series of tickboxes that allow users to select which data columns to include in their report (e.g., "Market ADR," "ADR Delta"). Phase 2 (Configurable API): Create a new, highly configurable backend endpoint (e.g., /api/generate-report ) that accepts the user's selections and dynamically builds a complex SQL query to generate the requested data. Phase 3 (Advanced Analytics): Implement the logic for advanced calculations, starting with "Market Revenue Adjusted for Hotel Size." This will involve creating new backend functions to normalize market data against the user's specific hotel capacity. Phase 4 (Exporting): Add functionality to export the generated reports. This should be done in stages, starting with the simplest format. Chunk 4a (CSV): Implement a server-side function to convert the JSON data to CSV format for download. Chunk 4b (PDF/Excel): Investigate and integrate server-side libraries (e.g., pdf-lib , exceljs ) to enable PDF and .xlsx exports. Phase 5 (Report Scheduler): Implement a system for users to schedule recurring reports. This is a major task that will require its own database tables to store schedules and a robust background job processor to run the reports and email them to users.**
3. **User Experience & Onboarding Objective: Improve the first-time user experience to increase adoption and reduce support requests. Key Tasks: Guided Product Tour: Implement a "first-login" guided tour that uses spotlights and tooltips to walk a new user through the key features of the dashboard. User Profile & Settings: Create a page where users can manage their account settings.**
4. **Access Control & Permissions Objective: Create different permission levels within the application to support both hotel staff and internal administrators. Key Tasks: Implement Role-Based Access Control (RBAC): Add a role column to the new users table (e.g., 'admin', 'user'). Superadmin View: The "Admin Panel" link in the sidebar should only be visible to users with the 'admin' role. The backend must also protect all admin API endpoints to ensure they can only be accessed by authenticated admin users.**

**Entry: Tuesday, July 8, 2025 - 9:09 AM CEST** **Objective: Document the final root cause analysis of the V2.0 data architecture and outline the strategic path forward pending external feedback.**

### **Part 1: Refresh Checkpoint & Final Root Cause Analysis**

A final review session was conducted to ensure a complete understanding of the project's data architecture and the precise nature of the OAuth invalid\_scope blocker.

**V1.0 Data Architecture (Legacy):**

* **Data Source:** The application's background scripts (initial-sync.js, daily-refresh.js) used a single, powerful master API Key stored in the .env file.
* **Permissions:** This master key belongs to our primary partner account, which has high-level permissions, including access to the data-insights API endpoints.
* **Data Flow:** The scripts successfully fetched data from the data-insights API and populated our own PostgreSQL database. The user-facing dashboard then read exclusively from our database, not the Cloudbeds API.

**V2.0 Data Architecture (Multi-Tenant):**

* **Data Source:** The V2.0 architecture is designed to use per-user OAuth 2.0 access tokens. Each new user who authorizes the application generates their own token.
* **The Core Problem:** The invalid\_scope error occurs because these newly generated tokens for new users do not have the data-insights permissions by default. This is a standard security practice by Cloudbeds, likely requiring a formal app certification to enable.
* **Conclusion:** The issue is not that the dashboard is broken, but that the **data synchronization process for new users** will fail. The V2.0 login flow was being tested on top of a V1.0 data set, which was the source of the confusion.

### **Part 2: Strategic Decision & Next Steps**

Based on the final analysis, the following strategic decisions have been made:

1. **External Blocker Confirmed:** We have confirmed the issue is an external dependency. An email has been sent to the Cloudbeds Integrations Team to clarify the process for activating the data-insights scopes and to resolve any potential confusion regarding our application credentials.
2. **No Code Revert:** We will **not** revert to the V1.0 codebase. The current V2.0 branch is stable for internal testing and represents significant progress towards our multi-tenancy goals. The existing OAuth flow works perfectly for the scopes that are currently enabled (read:user, read:hotel).
3. **Continue V2.0 Development:** Development will continue on the current V2.0 branch, and updates will continue to be deployed to Vercel. The live application is not considered "broken" for internal testing purposes.
4. **Continue Development with a Caveat:** Development on all V2.0 features, including those dependent on data-insights, can continue without restriction in our internal testing environment. This is possible because our background sync scripts still use our master API key, which has the necessary permissions. We acknowledge, however, that these specific features will not be functional for new, external users until the OAuth scope issue is resolved with Cloudbeds.

**Entry: Tuesday, July 8, 2025 - 9:45 AM CEST Objective: Commence work on the V2.0 "Advanced Reporting Module" while awaiting feedback on OAuth scope activation.**

**With the OAuth investigation concluded and strategic decisions made, development will now pivot to a key feature from the V2.0 roadmap that is unblocked for internal development.**

* **Immediate Task: Begin Phase 1 of the 'Advanced Reporting Module' - creating the foundational UI for a new /reports page.**
* **Plan: This will involve creating new files (/public/app/reports.html, /public/app/reports.js) and adding the necessary routing and navigation links to the main application shell.**
* **Focus: The initial work will focus purely on building the user interface for the report builder, including date pickers, granularity controls, and metric selection checkboxes, without connecting them to a backend API yet. This allows for parallel progress while the external dependency is being resolved.**