### **Project Refactor: Migrating to a Multi-PMS Adapter Architecture**

**Document Purpose:** This document outlines the plan and technical strategy for refactoring the Market Pulse backend from a single-PMS architecture to a modular, adapter-based system.

#### **1. Executive Summary: The "Why"**

Currently, our application is tightly coupled to the Cloudbeds API. Core business logic, data models, and authentication are all specifically designed to interact with Cloudbeds. While this has worked for our initial launch, it creates a significant technical barrier to future growth.

Our strategic goal is to support multiple Property Management Systems (PMS), with Mews being the next likely candidate. To do this efficiently, we must refactor our core application to be "PMS-agnostic." This means the main application shouldn't know or care where its data comes from. This refactor will introduce a "universal translator" layer, making future PMS integrations faster, safer, and more manageable.

#### **2. Proposed Refactoring Plan: The "How" and "Where"**

The refactoring will be executed in distinct, manageable phases.

##### **Step 1: Establish the Adapter Pattern**

The first step is to create the foundational structure for our new architecture. We will isolate all external API communication into dedicated "adapter" modules.

* **Create the Adapter Directory:** A new directory will be created at /api/adapters/. This will house a separate adapter for each PMS we integrate.
* **Build the Cloudbeds Adapter:** We will create the first adapter, /api/adapters/cloudbedsAdapter.js. This file will become the single source of truth for all interactions with the Cloudbeds API.
* **Prove the Concept:** To start, we will migrate a single, self-contained function (e.g., getNeighborhoodFromCoords) from its current location into the new cloudbedsAdapter.js. We will then update the part of the application that calls this function to point to the new adapter. This provides a low-risk way to validate the new structure.

##### **Step 2: Refactor Core Sync Logic**

Next, we will move the complex business logic out of our sync scripts and into the adapter.

* **Define a Canonical Data Model:** We will formally define a standard, internal data structure for key objects like DailyMetric and HotelDetails. The adapter's primary responsibility will be to return data in this exact format, regardless of how the source PMS structures it.
* **Refactor Sync Scripts:** The daily-refresh.js and initial-sync.js scripts will be refactored. All direct API call logic, data processing, and pagination handling will be moved into corresponding functions within the cloudbedsAdapter.js (e.g., adapter.getDailyMetrics(...), adapter.getHistoricalData(...)).
* **Simplify Core Logic:** After the refactor, the daily-refresh.js and initial-sync.js scripts will become much simpler. Their only jobs will be to load the appropriate adapter, call its high-level functions, and save the returned, already-standardized data to the database.

##### **Step 3: Abstract Authentication and Data Storage**

The final phase is to make our database schema and authentication middleware PMS-agnostic.

* **Generalize the Database Schema:** We will modify the users and user\_properties tables. Cloudbeds-specific columns (e.g., cloudbeds\_refresh\_token, override\_api\_key) will be replaced with generic columns like pms\_type (e.g., 'cloudbeds', 'mews') and pms\_credentials (a JSONB type that can hold different credential structures).
* **Abstract the Authentication Middleware:** The requireUserApi middleware will be updated. Instead of containing Cloudbeds-specific logic, it will first check the pms\_type for a user/property and then delegate the task of authenticating the request to the correct adapter.

#### **3. Benefits of this Approach**

This refactoring is a strategic investment in the application's future. Upon completion, our core codebase will be decoupled from any single vendor. Adding support for Mews (or any other PMS) will no longer be a complex, application-wide task. It will be a well-defined process of simply building a new adapter file (mewsAdapter.js) that conforms to the established internal standard, leaving the rest of the application untouched.