

Integrating 4D with Notion: Creating Pages and Databases Using the Notion API

By Al Mahdi Bakkali, Quality Support Engineer, 4D Inc.

Technical Note 25-08

Table of Contents

Table of Contents	2
Abstract.....	3
Introduction	3
Overview of 4D and Notion platforms.....	3
Use case: Real-Time Inventory Syncing from 4D to Notion	3
Benefits of integration: centralized knowledge, automation, and team collaboration	3
Prerequisites	4
Authentication with Notion API	4
Add a new Internal Integration	4
Registering a new Notion Page	5
Allowing access to a Notion Page	5
Obtaining the authentication key	5
Understanding Notion API Concepts	6
Pages vs. Database.....	6
Managing Session Data with Storage.....	7
Key API endpoints	7
Headers	7
Creating a page.....	8
Updating a page.....	8
Reading a database.....	9
Deleting a page	9
Mapping 4D Data to Notion Schema	10
Matching 4D variables to Notion Properties	10
Error Handling and Logging	10
Sample Demo: Business Application Workflow	11
Description of the Sample: Stock Management System in 4D.....	11
Stock Update in 4D and Notion.....	11
Qodly Studio Web App: A Seamless Integration with 4D.....	12
Presentation of the Qodly Studio web application	12
Key Features and Benefits	13
Ease of Integration	13
Code Reusability.....	13
Compatibility with 4D	14
Conclusion	14

Abstract

This technical note explores the integration of 4D, a powerful application development environment, with Notion, a modern workspace platform, using the Notion API. The guide demonstrates how to perform CRUD (Create, Read, Update, Delete) operations in Notion databases and pages programmatically from within a 4D application. By leveraging 4D's HTTP request capabilities and parsing tools, developers can seamlessly push structured business data—such as inventory—into Notion for tracking, collaboration, or analytics purposes. A sample business application will showcase real-world use, highlighting how completed transactions in 4D can automatically generate structured entries in Notion, transforming how teams visualize and manage operational data.

Introduction

The recent versions of 4D provide several features and commands dedicated to client web connections. The introduction of `HTTPRequest` class in earlier versions made significant improvement to the handling of HTTP requests and responses. This provides 4D with the ability to seamlessly connect and integrate with third-party services requested in the software market.

Overview of 4D and Notion platforms

Notion is a productivity software that has gained increasing popularity in the last years, not only in personal use for notetaking, but also in enterprise-level solutions for project management. Notion provides a versatile collaborative workspace that users can customize by adding databases, calendars, tables and many more. A user-friendly interface and the cross-platform availability of Notion have made it a choice for team collaboration where synchronized data plays a decisive role in successful team management.

Use case: Real-Time Inventory Syncing from 4D to Notion

The tech note aims to explore the integration of 4D with Notion to perform CRUD operations in Notion. This will allow the synchronization with a Notion database, with its corresponding pages, directly into 4D, and perform instant operation as mentioned above.

The use case presented in this tech note tries to mimic an inventory management system where customer-facing employees manage a store stock with Notion. These end users have a list of product items that update when new orders are received. In the backend, 4D plays the role of a stock management system of a storage facility, where cargos arrive to refill the stock. 4D users can update the stock of products and be reflected instantly in Notion.

Benefits of integration: centralized knowledge, automation, and team collaboration

Notion databases can present data in the form of a table, where each entry represents a record, or a “Page” in Notion. Each page may have several attributes that the end user can input.

The integration of 4D with Notion presented in this tech note can allow developers to centralize third-party data to visualize and modify records directly into 4D. In the use case presented, this integration will merge two business units by automating manual tasks of stock update in various software’s, encouraging business and team collaboration.

Prerequisites

- 4D 20R9
- A Notion integration created with Internal Integration Secret credentials
- Basic familiarity with JSON and REST APIs

Authentication with Notion API

Notion API provides a powerful toolkit of API endpoints to fully customize Notion workspaces, including pages, databases, users, comments, and more. The next section explores the different types of integrations provided by notion, as well as the set-up of an internal integration in Notion.

Notion provides two types of integrations to access the data depending on the use case. While the API endpoints remain the same, the authentication process is different. The key difference between the two integrations is the scope and access of the end-users.

Note: *Is the integration destined for public use? Is it the integration intended as an internal tool*

These are the type of questions a developer should ask when choosing the correct type of integration. Let us explore them in further detail:

- **Internal:** For private and internal workspaces.
- **Public:** For broad, public, and shareable workspaces.

As mentioned earlier, data access and permissions are the same for both integrations. This means that in both, developers have access to the same API endpoints. Read-write permissions will be configured in the Notion dashboard and are the same for the two types of integrations.

Add a new Internal Integration

For this tech note to mimic an enterprise solution, an internal integration to the Notion API is used.

Here is a detailed step-by-step tutorial to configure this type of integration:

1. Login to the notion account and go to <https://www.notion.so/profile/integrations>
2. Click on New Integration
3. Add the “Integration name”.
4. Choose the “Associated workspace” and in “Type” dropdown, select “Internal”. Optionally, upload a logo.

5. Click on Save
6. In the success alert, click on “Configure integration settings”

By following the step outlined above, the integration is now successfully created, and the browser has redirected to the integration settings page. This page is crucial for configuring and restricting access to the Notion API.

Registering a new Notion Page

The new integration does not have yet access to any pages yet, so an alert shows to add one in the “Access Tab”. Let’s add a new page to our Notion account from the template gallery:

1. Visit the template <https://www.notion.so/marketplace/templates/inventory-manager-database> and click on “Add”.
2. Wait a few seconds and a copy named “Inventory Manager” should appear in the left sidebar.
3. Click on “Inventory Manager” to display the new Notion database.

Allowing access to a Notion Page

Now we have our first page in Notion, it is time to add it to the integration.

Go back to the integration setting, visit the URL provided above or click on **“Settings” -> “Connections” -> “Develop or manage integrations”**. Let’s now set up access to the inventory manager page:

1. In the integration settings page, click on the “Access” tab and click on “Select pages”.
2. Search for the page or expand the collapsible menu “Private”.
3. Select “Inventory Manager” and click on “Update Access”.

Obtaining the authentication key

Back in the “Configuration tab”, in “Capabilities” the authorizations of the integration can be reviewed. For this tech note, the default values are kept.

In the same tab, the “Internal Integration Secret” displays a hidden code. Click on “Show” and “Copy” to get the internal integration secret. Example:

`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`

Note: Save the code and introduce it in the demo app when prompted.

This secret key is essential to the integration presented in this tech note as it is sent in each request header as an authorization bearer or token, as presented in the next sections.

Understanding Notion API Concepts

Now that the authorization token was successfully obtained, it is possible to make requests to the Notion API. The next section explores basic concepts in Notion and a selection of API endpoints to exemplify the integration of 4D.

Pages vs. Database

The concept of pages and databases can cause confusion to new users of Notion. A page is the main workplace in Notion where users can add a variety of graphical objects. Notion pages have properties used to capture structured information. Furthermore, each page is marked by a unique identifier.

On the other hand, Notion databases are a collection of pages which allow the developer to create and manipulate the structured data present in the pages.

In 4D jargon, a page is a record stored in a database, where the properties are table fields. In the case of use presented in this tech note, a table graphical object is used to hold individual product items, where each item corresponds to a single page, each with a unique differentiating ID.

Note: *Notion database and page ids are necessary when making API calls to notion. To get this id, open a database or page in Notion. In the right upper corner, click on “Share” and then “Copy Link”. Paste the content into a text editor and inspect the URL.*

Here are three examples of URLs that can be copied with the “Share” button:

1. https://www.notion.so/Smart-LED-Desk-Lamp-2454fd2ac9c581cd83c3c84786bbafc3?source=copy_link
2. <https://www.notion.so/Inventory-Manager-2454fd2ac9c5805a8e75e52a02b08e31>
3. <https://www.notion.so/Inventory-Manager-2454fd2ac9c5805a8e75e52a02b08e31?p=2454fd2ac9c581cd83c3c84786bbafc3&pm=s>

Example 1, corresponds to the URL of a page in Notion. The URL contains a 32 digit that corresponds to the id of that page.

In example 2, we can observe the same thing, but for a database.

Example 3, is a combination of the two, and shows first the id of the database followed by the id of the page.

In this tech note, the database id is obtained using this method. This id is sent as a parameter when making API request to Notion API and must be presented in a “8-4-4-12” format. Here is an example in 4D:

2454fd2a-c9c5-81dc-82bd-c1e22c6f0031

Note: *Save this code and introduce it in the demo app when prompted.*

Keeping track of the page id is also necessary, as updating existing pages will require their id. However, this id is obtained automatically in this tech note as it also serves to synchronize data entries between Notion and 4D.

Managing Session Data with Storage

The Storage object is a powerful tool in 4D for managing and sharing data across different parts of an application. It is a shared object that provides a simple and efficient way to store information that needs to persist for the duration of a session, such as configuration settings or user credentials.

In the demo application, the authorization bearer and the database id are prompted on database startup to be introduced by the user and later stored in the Storage object for later use through the application. Here is a snippet code that demonstrates this:

```
Use (Storage)
  If (Storage.notionConfig=NULL)
    ALERT("Please enter your Notion API credentials.")
    $bearerToken:=Request("Enter Notion Bearer Token:")
    $databaseId:=Request("Enter Notion Database ID:")
    Storage.notionConfig:=New shared object
  End if
End use

Use (Storage.notionConfig)
  Storage.notionConfig.bearerToken:="Bearer "+$bearerToken
  Storage.notionConfig.databaseId:=$databaseId
End use
```

Key API endpoints

As mentioned earlier in this document, this tech note aims to perform CRUD operation in a Notion database from 4D thanks to API calls. This section explores the main API endpoints that allow the achievement of these operations.

Headers

To successfully communicate with the Notion API, every request must include a set of specific headers. These headers provide crucial information, such as the type of data being sent and the necessary authentication token.

The Content-Type header specifies the format of the request body. For the Notion API, it is always application/json, indicating that the data payload is a JSON object. The Authorization header contains the Internal Integration Secret, prefixed with Bearer, which authenticates the request to the Notion workspace. Finally, the Notion-Version header is a required parameter that ensures the application is compatible with a specific version of the API.

Here is a sample 4D object with the headers for the Notion API:

```
$headers:=New object
$headers.Accept:="application/json"
$headers["Content-Type"]:= "application/json"
$headers.Authorization:="Bearer "+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"
$headers["Notion-Version"]:= "2022-06-28"
```

Creating a page

To create a new page in Notion, the “Create a page” API that corresponds to the following endpoint is used:

```
https://api.notion.com/v1/pages
```

This endpoint creates a child page inside a parent page or database, which ID is sent in the parent attribute. The endpoint has a series of parameters that must be sent to the request. Here is an example call in 4D:

```
$requestBody:=New object
$requestBody.parent:=New object("database_id"; $databaseId)
$requestBody.properties:=$properties
$dbRequest:=4D.HTTPRequest.new("https://api.notion.com/v1/pages"; {method: "POST"; headers:
$headers; body: $requestBody})
```

As observed in this code, two body parameters are sent in the request, namely the parent and properties parameters that are required when using this endpoint. In this tech note, the properties are matched with the input field of a 4D form.

Updating a page

To modify an existing page in Notion, the "Update page properties" API endpoint is used. This is a **PATCH** request that targets a specific page using its unique ID.

The endpoint URL is as follows, where {page_id} is the ID of the page to be updated:

```
https://api.notion.com/v1/pages/{page_id}
```

The request body of a PATCH call should only contain the properties to be changed. Notion's API is designed to handle partial updates, meaning sending the entire page's data is not necessary. This makes the update process simpler.

Below is an example of the 4D code for updating a page:

```
$dbRequest:=4D.HTTPRequest.new("https://api.notion.com/v1/pages/"+$pageId;
{method: "PATCH"; headers: $headers; body: $requestBody})
```


Reading a database

Retrieving data from a Notion database is done by making a **POST** request to the "Query a database" endpoint:

```
https://api.notion.com/v1/databases/{database_id}/query
```

This endpoint allows retrieval of all the pages (or records) within a specified database.

The response from this endpoint contains a collection of objects, where each object represents a page in the database. Each page object includes its unique ID and a properties object, which holds all the structured data for that page. This data includes all the properties of the Notion database, such as category, product name, stock level, and others.

In this tech note's use case, the database is read and these properties are mapped to a 4D collection, effectively synchronizing the data between the two platforms. The page's properties are parsed from the API response and used to populate a listbox in the 4D application, as seen in the following code snippet.

```
$item.category:=$page.properties.Category.select.name  
$item.dateAdded:=$page.properties["Date Added"].date.start  
$item.price:=$page.properties.Price.number
```

When reading a database from Notion, the id property of each page object is captured and stored in the 4D object as:

```
$item.id:=$page.id
```

This unique ID is crucial for performing future updates or deletions on that specific page from within the 4D application, as it acts as a key for synchronization.

Deleting a page

The Notion API doesn't have a specific **DELETE** endpoint for pages. Instead, pages are "deleted" by being archived. This is achieved by making a PATCH request to the same "Update page properties" endpoint used for updates.

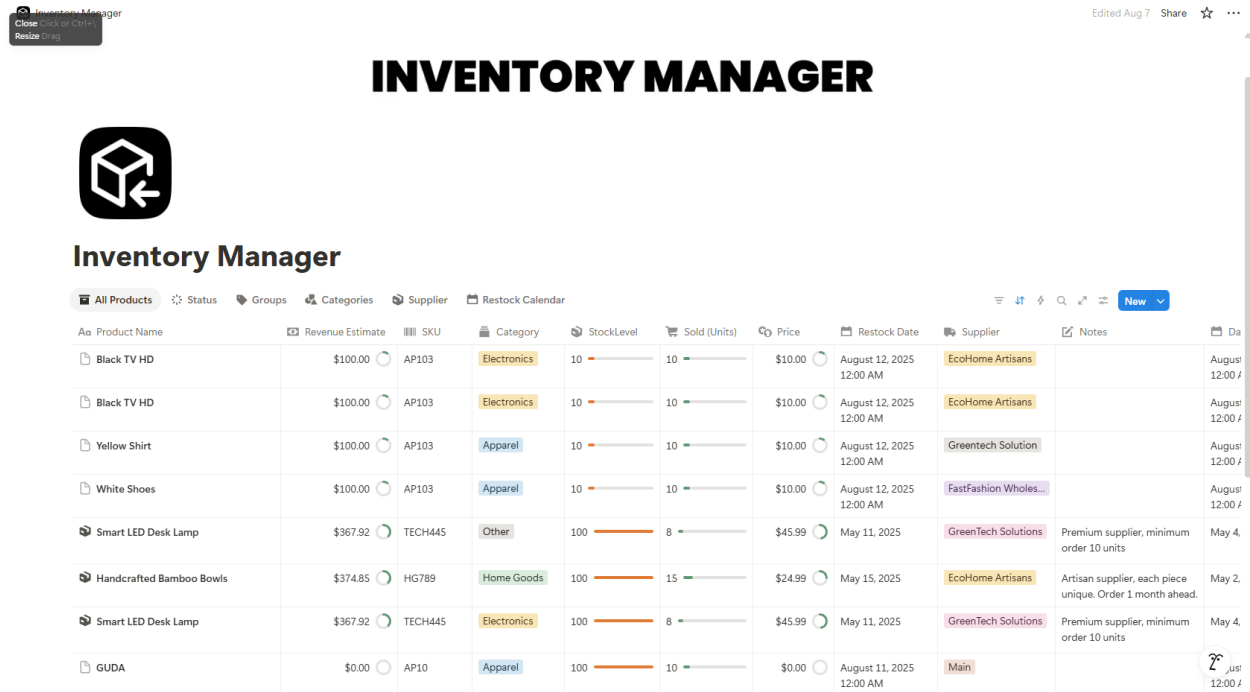
To archive a page, the archived property in the request body is set to true. This action removes the page from the active view in Notion but keeps it accessible in the archives, allowing for potential restoration later if needed.

Here is an example of how to archive a page in 4D:

```
$requestBody.archived:=True  
$dbRequest:=4D.HTTPRequest.new("https://api.notion.com/v1/pages/"+$pageId;  
{method: "PATCH"; headers: $headers; body: $requestBody})
```

Mapping 4D Data to Notion Schema

In the sample application, a collection of inventory items is worked with in 4D. Each item in the collection is an object with properties like **category**, **dateAdded**, **price**, **productName**, and **stockLevel**. This structure in 4D must correspond directly to the property schema in the Notion "Inventory Manager" database as shown in the picture below:



Product Name	Revenue Estimate	SKU	Category	StockLevel	Sold (Units)	Price	Restock Date	Supplier	Notes	Date
Black TV HD	\$100.00	AP103	Electronics	10	10	\$10.00	August 12, 2025 12:00 AM	EcoHome Artisans		August 12, 2025 12:00 AM
Black TV HD	\$100.00	AP103	Electronics	10	10	\$10.00	August 12, 2025 12:00 AM	EcoHome Artisans		August 12, 2025 12:00 AM
Yellow Shirt	\$100.00	AP103	Apparel	10	10	\$10.00	August 12, 2025 12:00 AM	Greentech Solution		August 12, 2025 12:00 AM
White Shoes	\$100.00	AP103	Apparel	10	10	\$10.00	August 12, 2025 12:00 AM	FastFashion Wholes...		August 12, 2025 12:00 AM
Smart LED Desk Lamp	\$367.92	TECH445	Other	100	8	\$45.99	May 11, 2025	GreenTech Solutions	Premium supplier, minimum order 10 units	May 4, 2025
Handcrafted Bamboo Bowls	\$374.85	HG789	Home Goods	100	15	\$24.99	May 15, 2025	EcoHome Artisans	Artisan supplier, each piece unique. Order 1 month ahead.	May 2, 2025
Smart LED Desk Lamp	\$367.92	TECH445	Electronics	100	8	\$45.99	May 11, 2025	GreenTech Solutions	Premium supplier, minimum order 10 units	May 4, 2025
GUDA	\$0.00	AP10	Apparel	100	10	\$0.00	August 11, 2025 12:00 AM	Main		August 11, 2025 12:00 AM

Matching 4D variables to Notion Properties

Each data type in 4D (e.g., text, number, date) must be mapped to a corresponding property type in Notion. For example:

- A 4D text field for **productName** maps to a Notion Title property.
- A 4D numeric field for **stockLevel** maps to a Notion Number property.
- A 4D date field for **dateAdded** maps to a Notion Date property.

This mapping is crucial for two-way synchronization. When a new item is created in 4D, its data is formatted into a JSON payload that Notion can understand. Conversely, when reading from Notion, the properties of the returned pages are parsed and assigned to the corresponding variables.

Error Handling and Logging

Robust error handling is essential for any API integration. Common issues include authentication errors, schema mismatches, and permission problems. In 4D, the status code of the HTTP response is a crucial indicator of success or failure.

- Dealing with Common API Errors

A successful API request will return a HTTP status code between 200 and 299. In 4D, this can be checked with a conditional statement like:

```
If (($dbRequest.response.status>=200) & ($dbRequest.response.status<300))
    // Successful response,
Else
    // use $dbRequest.response.statusText to get error text
End if
```

- **Authentication Errors:** An invalid or expired Authorization token will result in a **401** status code.
- **Schema Mismatch:** If the request body contains properties that do not match the Notion database's schema, the API will return a **400**-status code.
- **Permissions Issues:** A lack of necessary read or write permissions for a specific page or database will lead to a **403**-status code.

Sample Demo: Business Application Workflow

This sample desktop application demonstrates an inventory management system built with 4D that synchronizes with Notion. It is designed to mimic a real-world scenario where a 4D application manages a warehouse's stock, and a Notion database is used by customer-facing employees to track inventory.

Description of the Sample: Stock Management System in 4D

The 4D application acts as the backend for a storage facility. It features a user interface that allows warehouse staff to view and modify product stock levels. The application displays a list of all products with their current stock count, along with other details like SKU, price, and supplier.

Name	Revenue Estimate	Category	SKU	Date Added	Price	Restock Date	Sold Units	Status	Stock Level	Supplier
Black TV HD	\$100.00	Electronics	AP103	2025-08-11T23:00:00...	\$10.00	2025-08-11T23:00:00...	10	Out of Stock	10	EcoHome Artisans
Black TV HD	\$100.00	Electronics	AP103	2025-08-11T23:00:00...	\$10.00	2025-08-11T23:00:00...	10	Out of Stock	10	EcoHome Artisans
Yellow Shirt	\$100.00	Apparel	AP103	2025-08-11T23:00:00...	\$10.00	2025-08-11T23:00:00...	10	Low Stock	10	Greentech Solution
White Shoes	\$100.00	Apparel	AP103	2025-08-11T23:00:00...	\$10.00	2025-08-11T23:00:00...	10	Low Stock	10	FastFashion Wholes...
GUDA	\$0.00	Apparel	AP10	2025-08-10T23:00:00...	\$0.00	2025-08-10T23:00:00...	10	In Stock	100	Main
Handcrafted Bambo...	\$374.85	Home Goods	HGT89	2025-05-02	\$24.99	2025-05-15	15	In Stock	100	EcoHome Artisans
Smart LED Desk Lamp	\$367.92	Electronics	TECH445	2025-05-04	\$45.99	2025-05-11	8	In Stock	100	GreenTech Solutions
Smart LED Desk Lamp	\$367.92	Other	TECH445	2025-05-04	\$45.99	2025-05-11	8	In Stock	100	GreenTech Solutions

Stock Update in 4D and Notion

The synchronization process is triggered by a stock update event within the 4D application. When new inventory arrives, a warehouse employee updates the stock level of a product in the 4D system. This action automatically sends an API request to Notion, ensuring the data is immediately reflected in the front-end system used by sales teams. This eliminates manual data entry and reduces the risk of human error. In 4D, the method **getNotionDatabase** is used to refresh and synchronize the Listbox data each time an operation is performed.

Name	Revenue Estimate	Category	SKU	Date Added	Price	Restock Date	Sold Units	Status	Stock Level	Supplier
Black TV HD	\$100.00	Electronics	AP103	2025-08-11T23:00:00...	\$10.00	2025-08-11T23:00:00...	10	Out of Stock	10	EcoHome Artisans
Black TV HD	\$100.00	Electronics	AP103	2025-08-11T23:00:00...	\$10.00	2025-08-11T23:00:00...	10	Out of Stock	10	EcoHome Artisans
Yellow Shirt	\$100.00	Apparel	AP103	2025-08-11T23:00:00...	\$10.00	2025-08-11T23:00:00...	10	Low Stock	10	Greentech Solution
White Shoes	\$100.00	Apparel	AP103	2025-08-11T23:00:00...	\$10.00	2025-08-11T23:00:00...	10	Low Stock	10	FastFashion Wholes...
GUDA	\$0.00	Apparel	AP10	2025-08-10T23:00:00...	\$0.00	2025-08-10T23:00:00...	10	In Stock	100	Main
Handcrafted Bambo...	\$374.85	Home Goods	HG789	2025-05-02	\$24.99	2025-05-15	15	In Stock	100	EcoHome Artisans
Smart LED Desk Lamp	\$367.92	Electronics	TECH445	2025-05-04	\$45.99	2025-05-11	8	In Stock	100	GreenTech Solutions
Smart LED Desk Lamp	\$367.92	Other	TECH445	2025-05-04	\$45.99	2025-05-11	8	In Stock	100	GreenTech Solutions

Add New Item
Delete Item
Update Stock
Get Inventory

When the update button is clicked the code below is executed in 4D to update the stock levels:

```

$newStockValue:=100
$headers:=New object
$headers.Accept:="application/json"
$headers["Content-Type"]:= "application/json"
//$headers.Authorization:="Bearer
"+"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"
$headers["Notion-Version"]:= "2022-06-28"

Use (Storage)
    $headers.Authorization:=Storage.notionConfig.bearerToken
End use
$requestBody:=New object
$requestBody.properties:=New object
$requestBody.properties.StockLevel:=New object("number"; $newStockValue)
$requestBody.properties.Status:=New object("status"; New object("name"; "In Stock"))
$dbRequest:=4D.HttpRequest.new("https://api.notion.com/v1/pages/"+$pageId; {method:
"PATCH"; headers: $headers; body: $requestBody})
$dbRequest.wait(30)

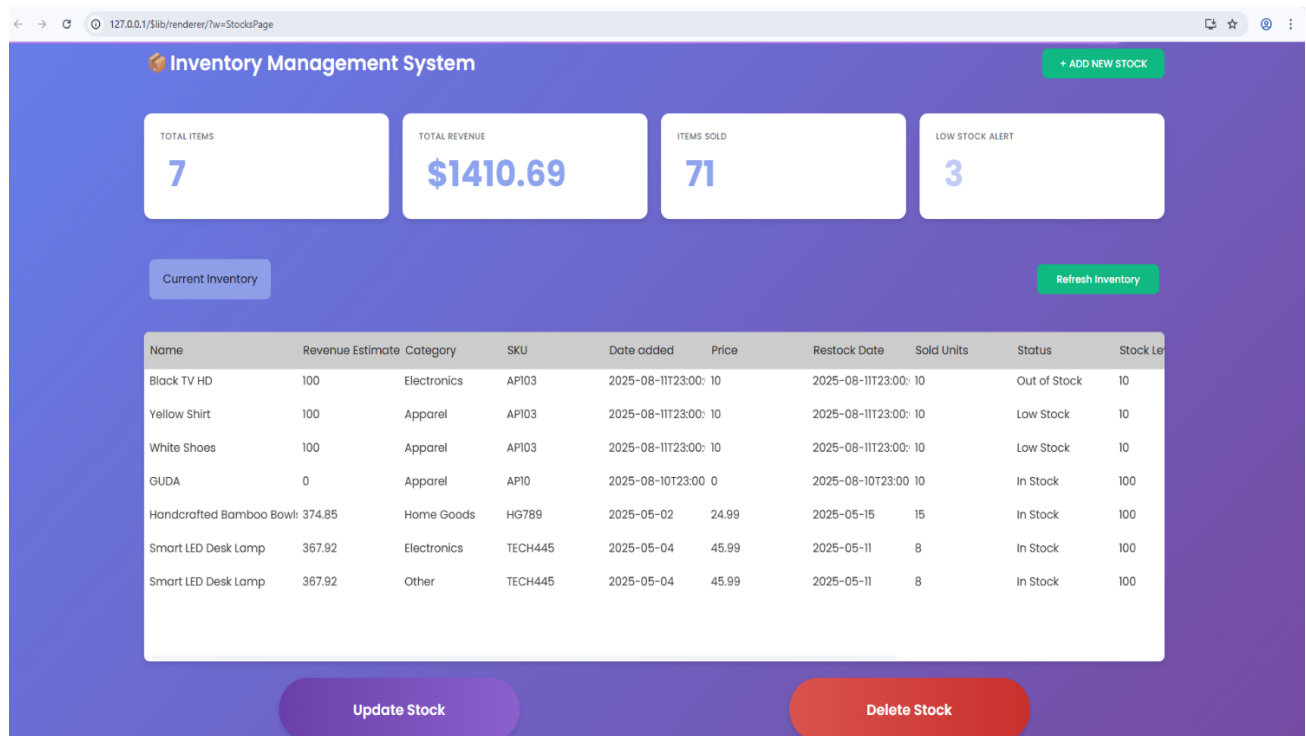
```

Qodly Studio Web App: A Seamless Integration with 4D

Presentation of the Qodly Studio web application

The Qodly Studio web application is an inventory management system with a clean, modern user interface. This web app is a direct representation of the functionality available in the desktop 4D demo, showcasing the seamless integration and powerful features that

Qodly Studio provides. The application displays key metrics such as Total Items, Total Revenue, Items Sold, and a Low Stock Alert, offering a comprehensive overview of the inventory status.



Here are some instructions to get started with Qodly Studio and try the demo app:

- **Check Prerequisites:** Before beginning, ensure to have all the necessary components by checking the documentation at <https://developer.4d.com/docs/WebServer/qodly-studio>
- **Open Qodly Studio:** In 4D, navigate the Design menu and click on "Qodly Studio...". This action will open Qodly Studio in the default web browser.
- **Launch and Preview:** Make sure that the 4D Web Server is launched. Once it is running, click on the Preview button within Qodly Studio.

Key Features and Benefits

Ease of Integration

One of the most significant advantages of Qodly Studio is its ease of integration with the 4D environment. As per the instructions, the integration process is straightforward. This direct link allows developers to leverage their existing 4D data and business logic without complex configuration or setup. The web application can be previewed instantly in a browser, enabling rapid development and testing cycles.

Code Reusability

Qodly Studio promotes code reusability by allowing developers to build web applications that can interact with the same backend business logic and data models used in their 4D applications. Although Qodly Studio uses its own scripting language, the underlying architecture ensures that data and rules defined in 4D are accessible and can be utilized within the web app. This means that a function or a method written in 4D to manage inventory can be called from the Qodly application, eliminating the need to rewrite the same logic for a different platform. This approach not only saves time but also ensures consistency across different applications. In the demo application, the Qodly method are available under “Classes” in the “Datastore”.

Name	Revenue Estimate	Category
White Shoes	100	Apparel
Test1234	180	Home Goods
TEST1111	0	Home Goods
test1	7200	Electronics
QLZO	100	Apparel
LOL	0	Apparel
ALO	0	Apparel
GUDA	0	Apparel
Handcrafted Bamboo Bowl	374.85	Home Goods

Compatibility with 4D

Qodly Studio is designed for deep compatibility with 4D. This synergy allows developers to extend the functionality of their 4D applications to the web easily. The inventory management system exemplifies this compatibility: actions like adding new stock, updating stock, or deleting stock are all handled through the web interface but trigger the corresponding business logic and database operations defined in the 4D application. This two-way communication ensures that data remains synchronized and that the web app remains a true extension of the core 4D system.

Conclusion

Integrating 4D with Notion through the Notion API provides a powerful way to bridge two distinct platforms, combining the robust application development capabilities of 4D with the flexible, collaborative workspace of Notion. By following the steps outlined in this technical note, developers can integrate 4D with Notion and perform other API calls under the same principle. The presented use case of an inventory management system demonstrates how 4D can connect and synchronize with third-party applications.