Technical Document

|  |  |
| --- | --- |
| Scope | Logging the BG User interaction inside the  7digital and BG application |
| Version | 1.0.0 |
| Date | 26-01-2023 |

2023

Table of Contents

[Overview 2](#_Toc124280423)

[Logging User Activity inside 7digitalApp 3](#_Toc124280424)

[Logging](#_Toc124280425) User Activity outside 7digitalApp

[Implementation 6](#_Toc124280426)

[API key generation 7](#_Toc124280427)

[API Endpoint 7](#_Toc124280427)

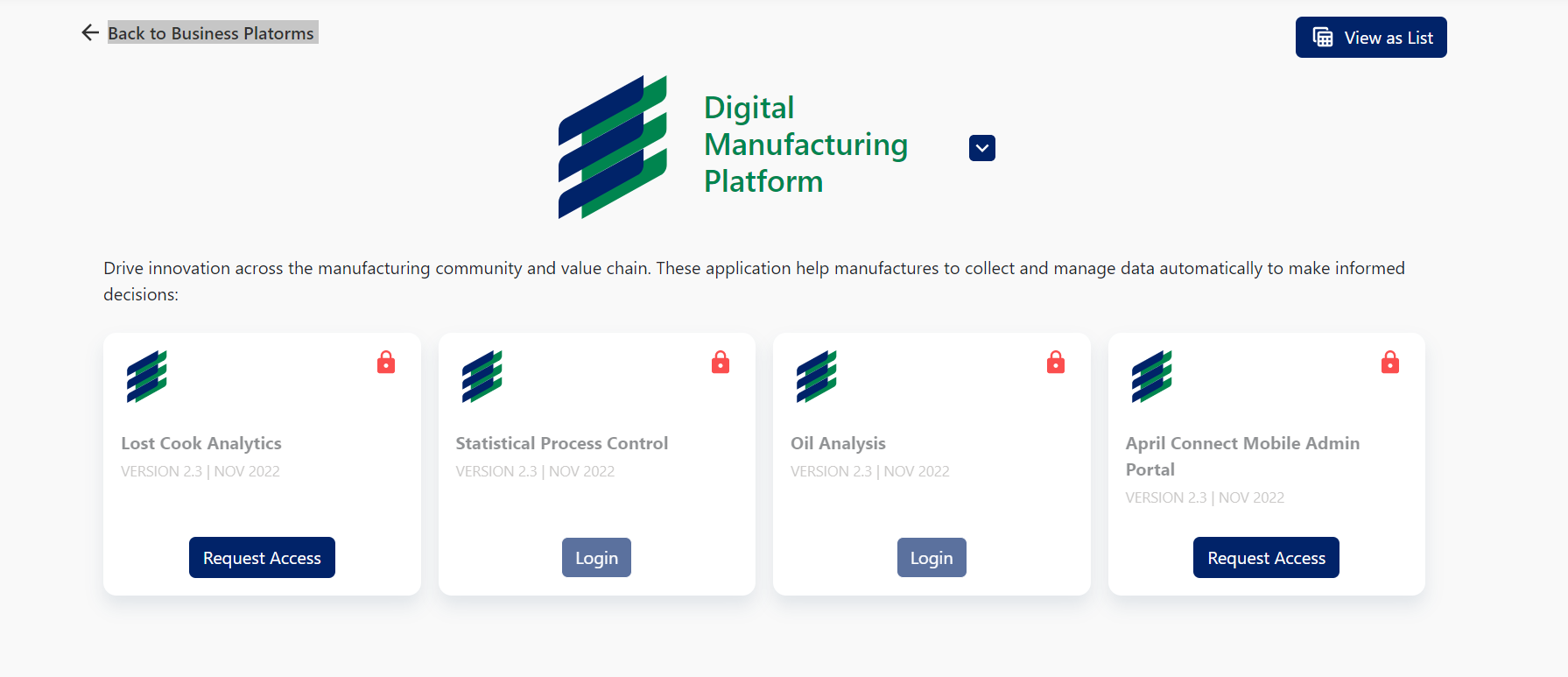
# Overview

Through this functionality we are trying the monitor the user interaction within the 7dipital App and Application. And display the metrics on admin dashboard through charts and graphs

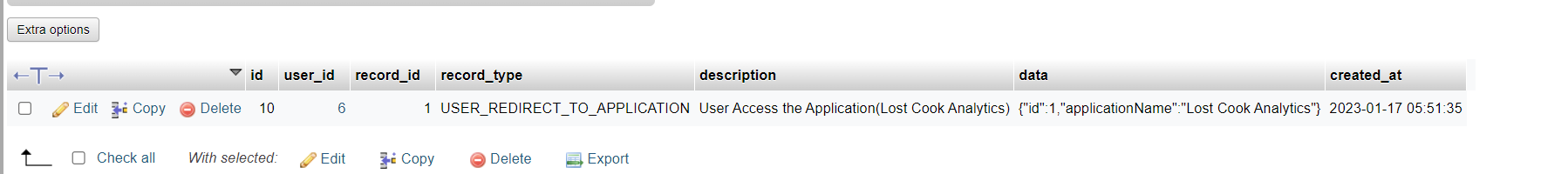
# Monitor User Inside 7digitalapp

To be able to display 7digitalapp application usage We will log user action whenever BG-User uses the 7digitalApp platform to access the other BG Application

**Figure 1.** Application Page



When the User clicks on the login page we are logging the an event with following info



user\_id: BG USER ID,

record\_id: Application Id

record type: Unique event name for application redirect

# Logging user activity outside 7digitalpp

Through this feature we are monitor the user activity inside the BG Application

We will log following information

* Module Name or functionality user is currently accessing
* Full URL path at which the event occured
* Timestamp at which the event occurred
* User : who performed the action
* Application Id : on which action is performed

## API Key Generation

For the Communication between the BG application and 7digitalApp we will provide the API. Each application will be provided with their unique API key to access the API endpoint. API KEY will serve as authentication token as well as unique identifier to for application

API Keys are simple to use, they’re short, static, and don’t expire unless revoked. They provide an easy way for multiple services to communicate.

Since the API key itself is an identity by which to identify the application or the user, it needs to be unique, random and non-guessable. API keys that are generated must also use Alphanumeric and special characters.

An example of such an API key is BoU5s9.TvLQfzH1/dC1H1cfjDWGHAc85vHD0UHAPSRNZpfw1PQ=

## Secure API Key Storage

The reason we need to store API keys is to make sure that the API key in the request is valid and issued by us (just like a password).

We don’t need to know the raw API key, but just need to validate that the key is correct. So instead of storing the key in plain text (bad) or encrypting it, we should store it as a hashed value within our database.

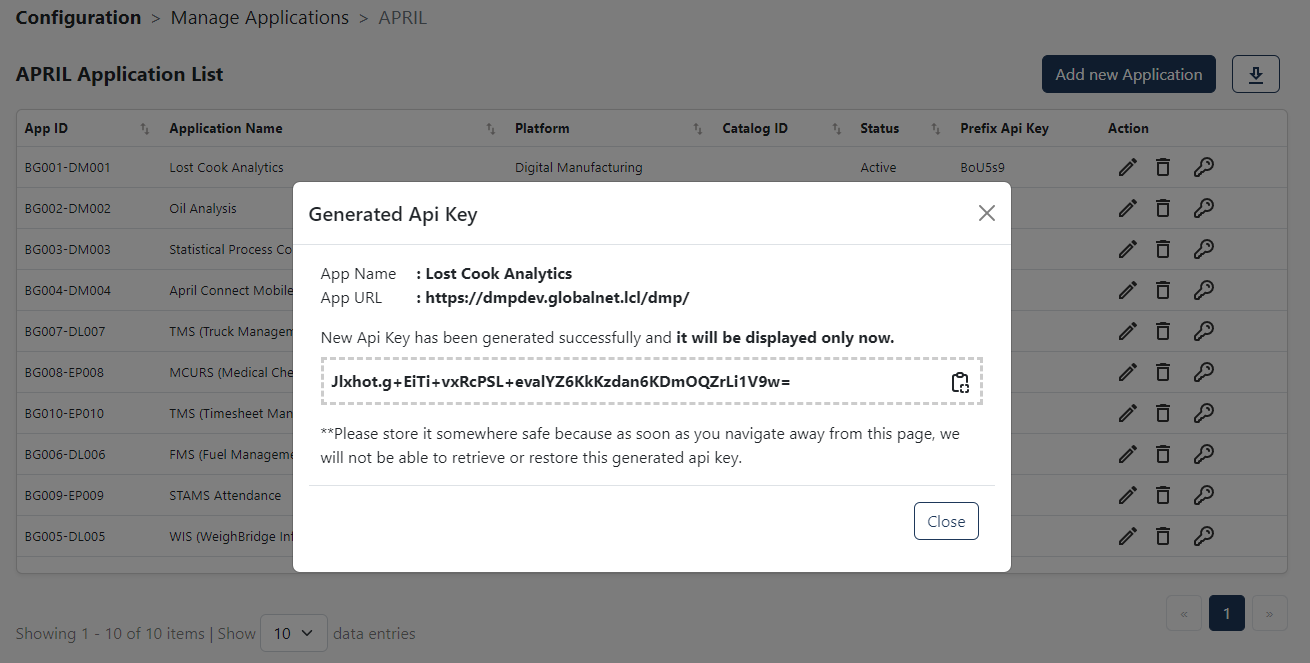
A hashed value means that even if someone gains unauthorised access to our database, no API keys are leaked and it’s all safe. The end user would send the raw API key in each API request, and we can validate it by hashing the API key in the request and compare the hashed key with the hash stored within our database.

BoU5s9.TvLQfzH1/dC1H1cfjDWGHAc85vHD0UHAPSRNZpfw1PQ=

In the code above, the primary key will be a combination of the prefix and the hash of the API key {prefix}.{hash\_of\_whole\_api\_key}.

## Presenting the API Keys to users

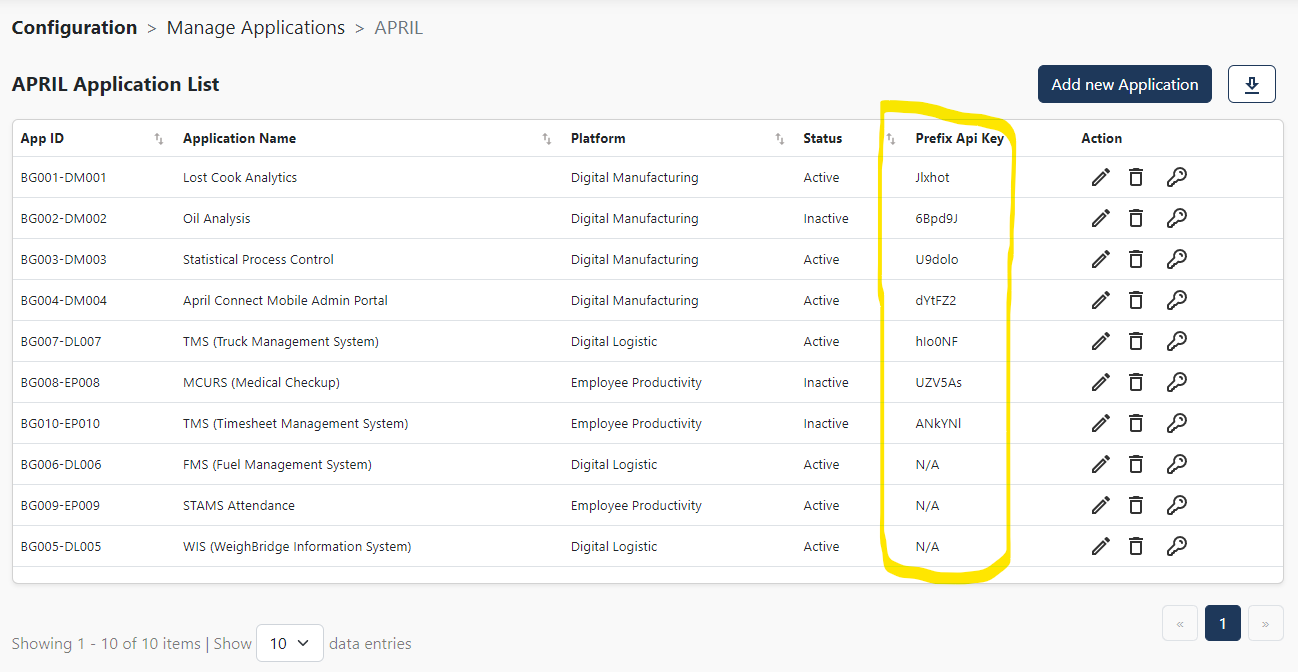
Since we don’t store the original API key, we can show it only once to the user, at the time of creation. So be sure to alert users that it cannot be retrieved again, and they need to generate a new token if they forget to copy the API key and store it safely. We can do something like this:



## How user can identify a generated API Key later

Another problem is how users identify the right API key in your console if they need to edit or revoke it. This can be solved by adding a prefix to the API key. Notice in the picture above **the first 6 characters (that’s our prefix),**separated by the dot.

Now you can store this prefix in the database and display it in the console so users are able to quickly identify the right API key entry, like this:



## Implementation

For the API Key generation we can create from shell script that can be executed from the backend or user can also create from the UI.

The API key will have lifetime validity until we create a new API key for that particular application.

For the API Key we use several function from built-in nodejs packages “crypto”:

* Generate unique, random and non-guessable prefix and key
* Generate hashed value for that api key
* Validate Function by hashing the API key in the request and compare the hashed key with the hash stored within our database

Table Schema : application\_api\_keys

|  |  |  |
| --- | --- | --- |
| application\_id | integer | Primary Key for this Table |
| prefix | VARCHAR(200) |  |
| key | VARCHAR(200) |  |
| created\_at | timestamp |  |
|  |  |  |

## API Endpoint

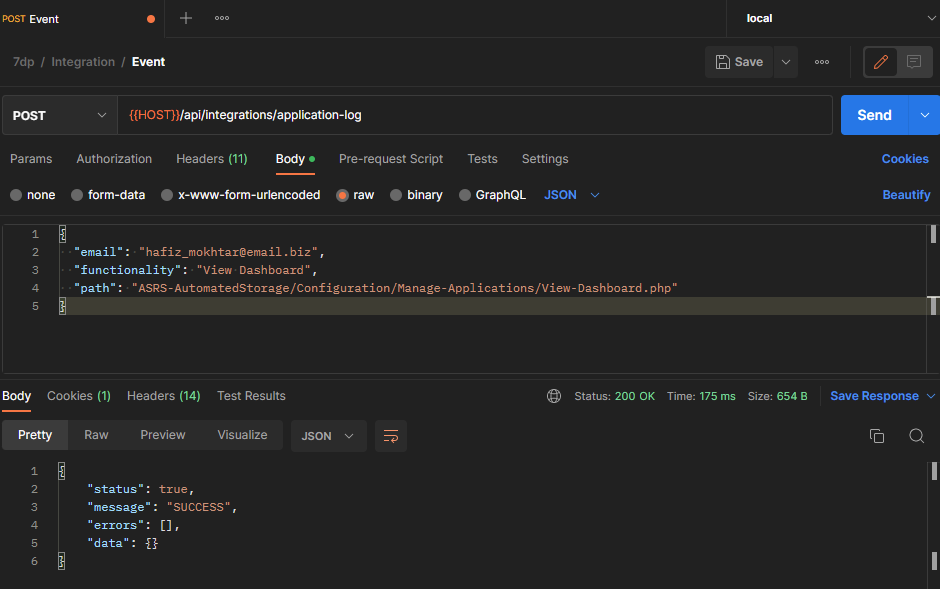
API definition

|  |  |
| --- | --- |
| END POINT | /api/integrations/application-log |
| Content-Type | Application/json |
| Header | x-api-key: {Unique API key for the application} |
| Payload | |  |  | | --- | --- | | email | User Email act as unique identifier for the User. (ex: [hafiz\_mokhtar@email.biz](mailto:hafiz_mokhtar@email.biz)) | | functionality | Unique module name inside the application. TDB by the application owner. (ex: “View Dashboard”) | | path | Full url path which event occurred.  (ex: “ASRS-AutomatedStorage/Configuration/Manage-Applications/View-Dashboard.php”) | |

Table Schema :application\_logs

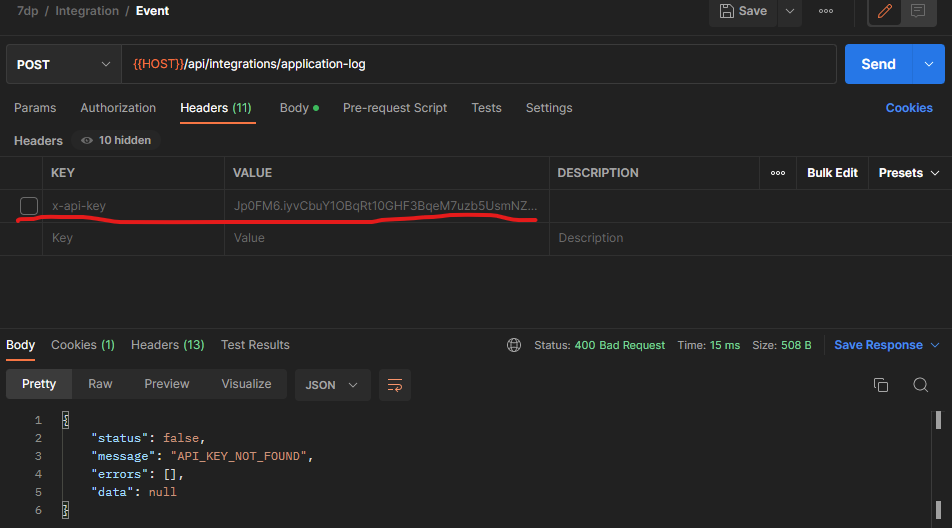
|  |  |  |
| --- | --- | --- |
| id | Integer | Primary key |
| application\_id | Integer |  |
| user\_id | Integer |  |
| functionality | VARCHAR(200) |  |
| path | VARCHAR(200) |  |
| created\_at | timestamp |  |

**Success Response**

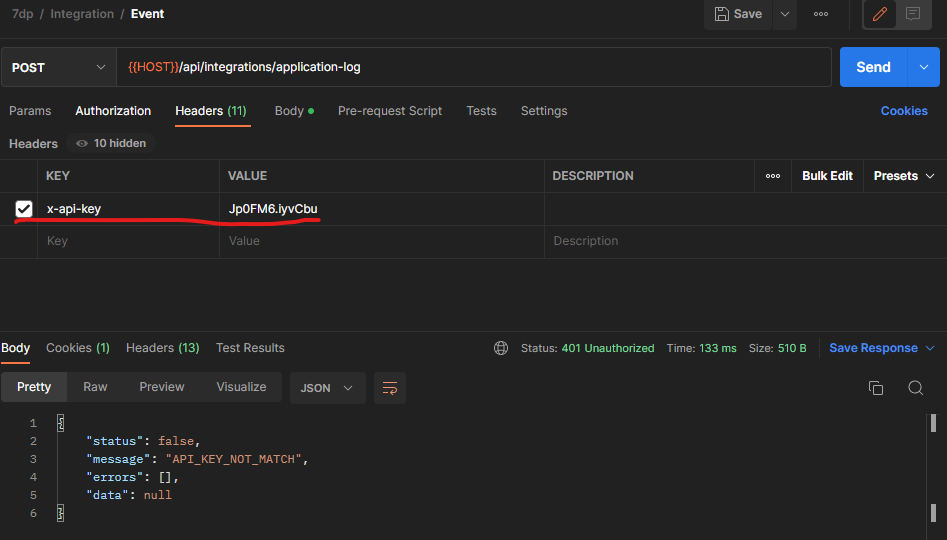
****

**Failed Response**

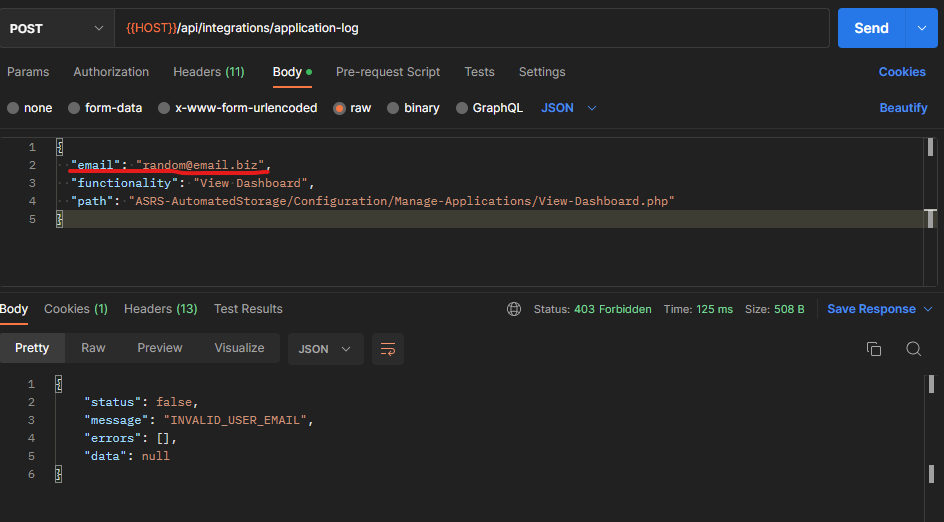
1. API\_KEY\_NOT\_FOUND



1. API\_KEY\_NOT\_MATCH



1. INVALID\_USER\_EMAIL



1. INVALID\_REQUEST

