Open Innovation Digital Platform and Fablabs for Collaborative Design and Production of personalised/customised FMCG

DT-FOF-05-2019 (870148)

User Manual Report

Deliverable ID	D5.3	**Version**	V1
Deliverable name	User Manual Report		
Lead beneficiary	RDIUP		
Contributors	Dah DIARRA (RDIUP), Habib NASSER(RDIUP)		
Due date	28-02-2021		
Date of final version	27-02-2021		
Dissemination level	PU		
Deliverable type	R: Document, report (excluding the periodic and final reports)		
Document approval	Chandana Ratnayake	27.02.2021	[illegible]

The DIY4U project has received funding from the European Union's Horizon 2020 research and innovation programme under GA No. 870148

Deliverable details

Deliverable ID: D 5.3

Deliverable name: User Manual Report

Responsible partner: RDIUP

Contributors: Dah DIARRA (RDIUP) and Habib NASSER (RDIUP)

Reviewed by: Alexandru BUTEAN (EFF), Mark TAYLOR (CPI)

Due date: 28/02/2021

Delivery date: 27/02/2021

Version	**Authors**	**Date**	**Status**
First draft	Dah DIARRA (RDIUP), Habib NASSER (RDIUP)	18-02-2021	To be reviewed
Review	Alexandru BUTEAN (EFF), Mark TAYLOR (CPI)	23-02-2021	To be improved
Final Version	Dah DIARRA (RDIUP), Habib NASSER (RDIUP)	26-02-2021	To be submitted

Table of Contents

Abbreviation:

API: Application Programming Interface	**JWT**: JSON Web Token
EF: Extension Functionalities	**OIP**: Open Innovation Platform
FMCG: Fast Moving Consumer Goods	**P&G**: Procter and Gamble
HTTP: Hypertext Transfer Protocol	**REST**: Representational State Transfer
JSON: JavaScript Object Notation	**URL**: Uniform Resource Locator

1. Executive Summary

DIY4U has the ambition to be a leader in the digital customization of FMCG. To reach this goal, WP5 proposes Extension Functionalities (EFs) to better understand the behaviors and personalised needs of consumers, extract meaningful information and recommend formula in an effective and secure manner. The data analytics API will interact with other modules in the an Open Innovation Platform (OIP) which explains the importance of a user manual for key technical partners to facilitate their development activities in WP4 and WP5. The user manual contains the following 6 sections:

1. An introduction: will be defined to deliver a technical overview about the user manual.
2. API description: An introductive documentation and graphics are provided to present the data analytics structure and different layers.
3. Authentication and authorization: In this report, a method will be presented and detailed to support technical partners use and test their linked modules.

4. Resources: The main resources will be provided and presented to define each parameter and attribute and understand the associations between them.
5. Management of resources: includes the HTTP or HTTPs requests to be used by other WPs to interact with our data analytics modules.
6. A brief and clear conclusion about the user manual and further perspectives will be reported.

This report will be utilized as a guide in WP4, WP5 and open calls to support key stakeholders developing their tools.

2. Introduction:

This Deliverable (D5.3) aims to create an intuitive, and appealing user manual. It consists of the definition of a guide to support DIY4U's partners using and interacting with our Back-end modules developed in the Task 5.2. This report will first introduce the structure, the architecture and the different layers of our modules. Then, it will present the authentication methods proposed in this DIY4U's API, the postman tool and all available resources. The User Manual clearly explains how the resources will be managed to help partners learn how to use the data analytics module, and provide a short description of each resource and linked request. The User Manual will be written using technical and non-technical terminology and will include the key features and functions developed in D5.2 "Data analytics application and results report".

3. API description

DIY4U EFs is an API (Application Programming Interface) built around REST (Representational State Transfer) specifications. The application has predefined URLs for its resources accessed by the client application, It accepts JSON-encoded format in the request body and returns response in JSON format. These predefined URLs are used over HTTPS or HTTP with http verbs (GET, POST, PUT, DELETE and PATCH) to interact with the API. As the API has restricted access and secure, authentication is required. A user can generate an authorization token in order to discover and manage the API features. Read the authentication section for deep understanding.

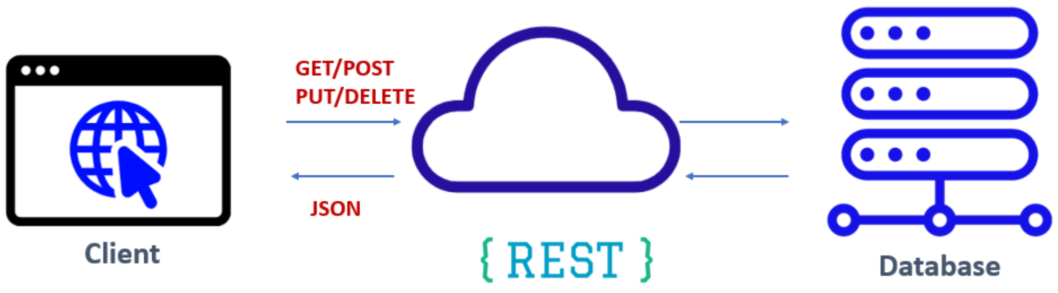
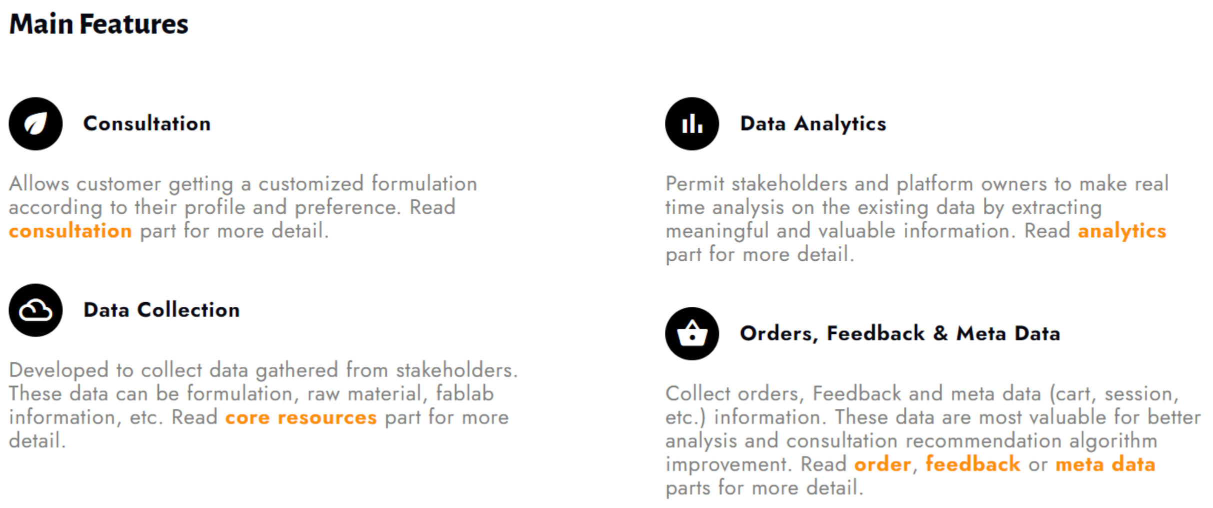


Figure 1: Overview of our solution

The database of the APIs was developed using MongoDB which is a flexible and scalable database management system. Data is structured in the database in the form of many collections and each collection contains many documents of the same structure. Each single collection of the database is considered as a resource that needs to be managed during the application lifecycle. The resource section will describe the contains of each resource and the management of resources describes how to manage it during the resources lifecycle. The Figure 2 shows the main features (Consultation, Data collection, Data analytics and order/feedback) of our tool. Moreover, RDIUP has defined a web-based documentation interface to introduce our solutions https://diy4u-frontend.herokuapp.com/docs/api/introduction



Main Features

Consultation

Allows customer getting a customized formulation according to their profile and preference. Read **consultation** part for more detail.

Data Analytics

Permit stakeholders and platform owners to make real time analysis on the existing data by extracting meaningful and valuable information. Read **analytics** part for more detail.

Data Collection

Developed to collect data gathered from stakeholders. These data can be formulation, raw material, fablab information, etc. Read **core resources** part for more detail.

Orders, Feedback & Meta Data

Collect orders, Feedback and meta data (cart, session, etc.) information. These data are most valuable for better analysis and consultation recommendation algorithm improvement. Read **order**, **feedback** or **meta data** parts for more detail.

Figure 2 : The main features of the data analytics module

The following subsections present the main structure and layers (presentation, application and data access layer) of our Back-end API.

Modules Interactions: As Django has its own style of architecture, it provides an effective way to define many small applications within the project. these small applications act as one module in the present architecture. The different modules can interact between them by exchange data between each other. The core module is the one that provide implementation of primary resource. It is used by all modules to access some implementation. This module does not required other module to work. Consultation, Analytics, and orders are datasets that provide specific of the application. These modules focus on their own features and they are totally dependent to the core module.

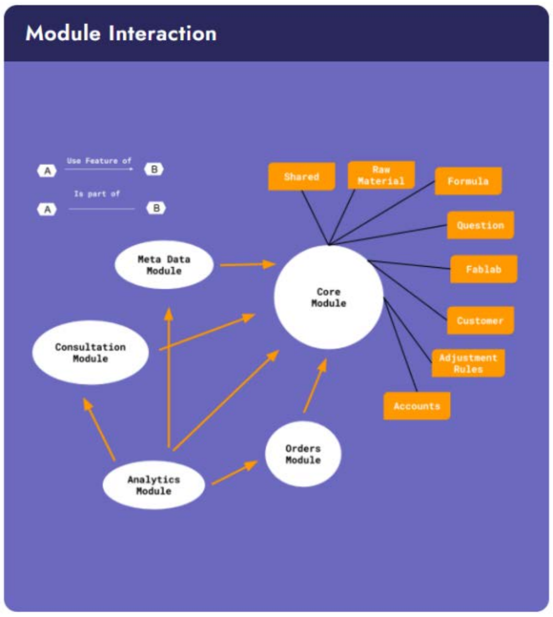


Figure 3 : Interaction between modules

Module Architecture: The architecture is very important to build a great software. There are many type of architecture developed in the literature such as Multitier, Monolithic, Microservices, Event Driven etc.. The three-tier architecture which is a Multitier architecture was the one used in this project. It has three main layers (Presentation, application, data access) where communication goes from top layer to bottom layer for incoming request and reverse direction for answer. Each layer manages a specific and different tasks of the application.

Presentation layer is the top level layer which is responsible for the interaction with users or other program that interact with the app. For REST API all related code in this layer are focusing on routing incoming request and call the appropriate application layer code to get result and return this result as request response. **Application** layer is the middle level layer, it contain the business logic of the app. These business logic are class or functions that contain implementation of the application features. It is used by the presentation layer to treat request and use the data access layer to get data. **Data Access** layer is the bottom level layer that contains code for reading and writing in the data base.

Presentation Layer: Since our app is REST API, this layer is responsible for request routing, respond request, error handling, authentication and authorization. Incoming requests are first handle by the built-in **django router** , which call the appropriate view class and method based on pre-configure URL and the incoming request (method, URL, parameters etc..). For example POST https://diy4u-efs.herokuapp.com/ is different to PUT https://diy4u-efs.herokuapp.com/. When the corresponding view is called by the router, it checks whether the request is **authenticate** and the user has **authorization** access. It returns an authentication error if the request is not authenticate or user is not authorized, otherwise it calls application layer to get result of the treatment of the request and returns it as response. The application layer can raise an **error**, so it return a well formatted error response.

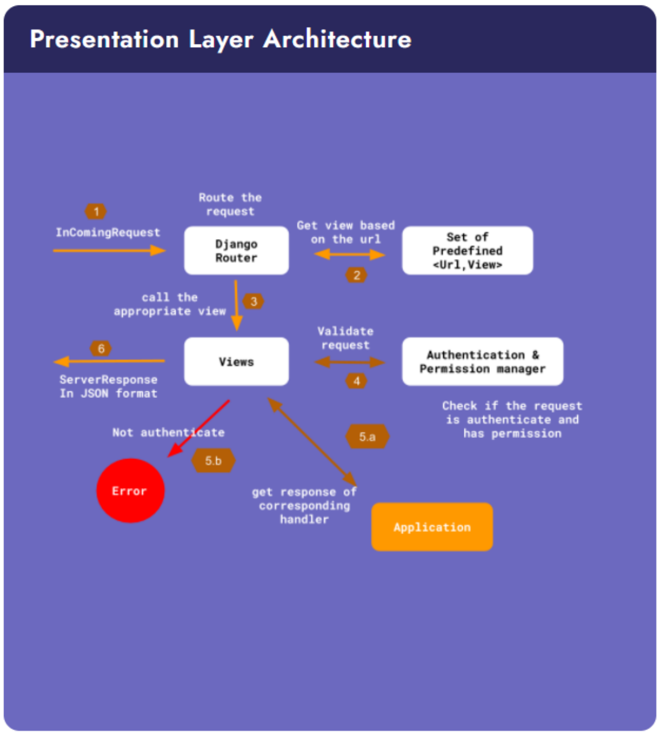
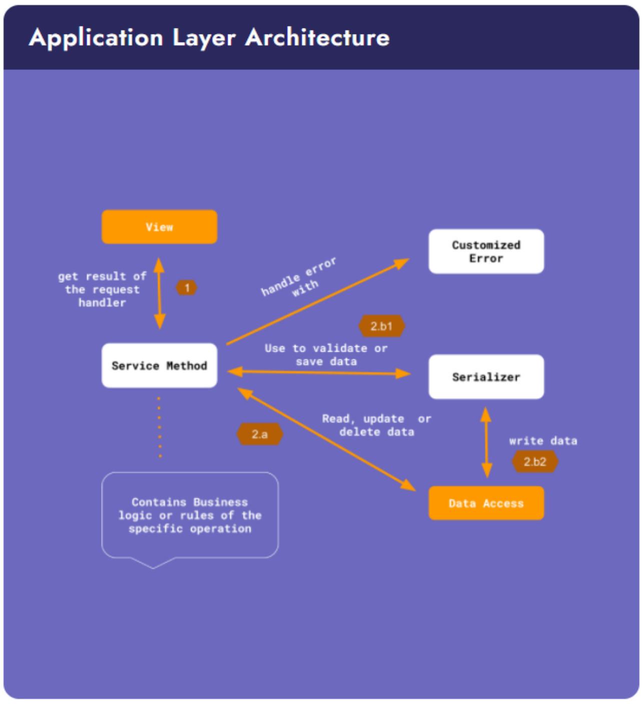

Presentation Layer Architecture
1
InComingRequest
Route the request
Django Router
Get view based on the url
2
Set of Predefined <Url,View>
call the appropriate view
3
6
ServerResponse In JSON format
Views
Validate request
4
Authentication & Permission manager
Check if the request is authenticate and has permission
Not authenticate
5.b
5.a
Error
get response of corresponding handler
Application

Figure 4 : Presentation Layer


Application Layer Architecture
View
get result of the request handler
1
handle error with
Customized Error
2.b1
Use to validate or save data
Service Method
Serializer
Read, update or delete data
2.a
write data
2.b2
Contains Business logic or rules of the specific operation
Data Access

Application Layer: This layer is the one that describes the behaviour of the system by implementing all features or services of the system or the module. It ensures that all required policies are respected and validated during service or feature execution. As this layer provides a set of service or features, they are imported and used inside the presentation layer. During the service execution, they can access to the data access layer to get or save data. It also handles its own execution error and provides one unique customized error to the view. **Serializers** are also used by this layer to validate data (this is more specific to Django technology), save data and convert data from python object to JSON format and vice-versa. Utils class and functions are also part of the layer to reduce service class and function size.

Figure 5 : Application Layer architecture

Data Access Layer: This layer is responsible for handling database access for write and reading data. It Contains class and method that handle write and read operation. As there no business policies in this layer to check data integrity, this layer is only available to the application layer which guaranteed the integrity of data.
The business key object are defined as model that extends **Mongoengine** package document class which provide implementation of write and read operation on the database. Each model should contain attribute and their type, mongoengine will automatically create collections and documents in database when needed for those model. The Database used in the DIY4U project **MongoDB** which is a document based database system. This type of database allows to store an object in **BSON** format. It is hosted in **MongoDB Atlas** : a cloud service for database hosting.

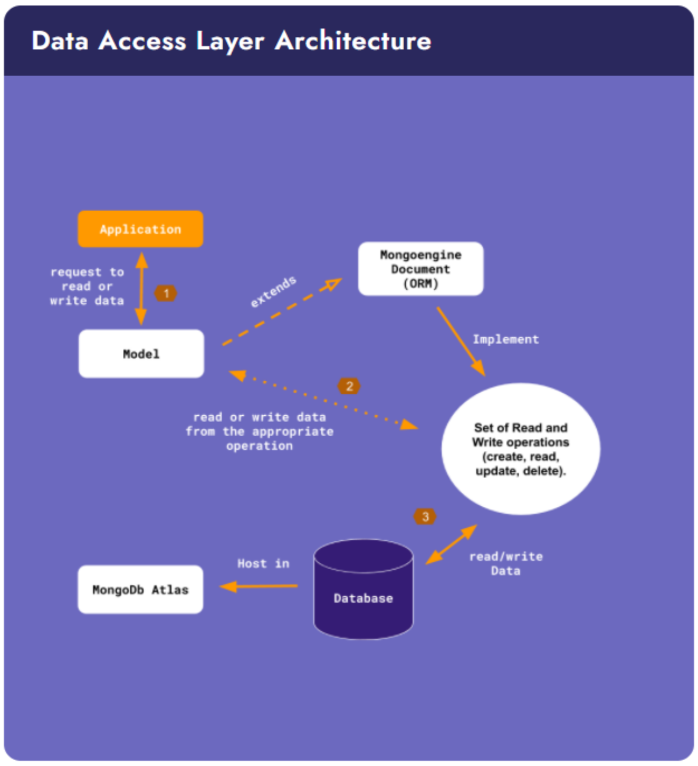

Data Access Layer Architecture
Application
request to read or write data
1
Model
extends
Mongoengine Document (ORM)
Implement
2
read or write data from the appropriate operation
Set of Read and Write operations (create, read, update, delete).
3
read/write Data
Database
Host in
MongoDb Atlas

Figure 6: Data access layer architecture

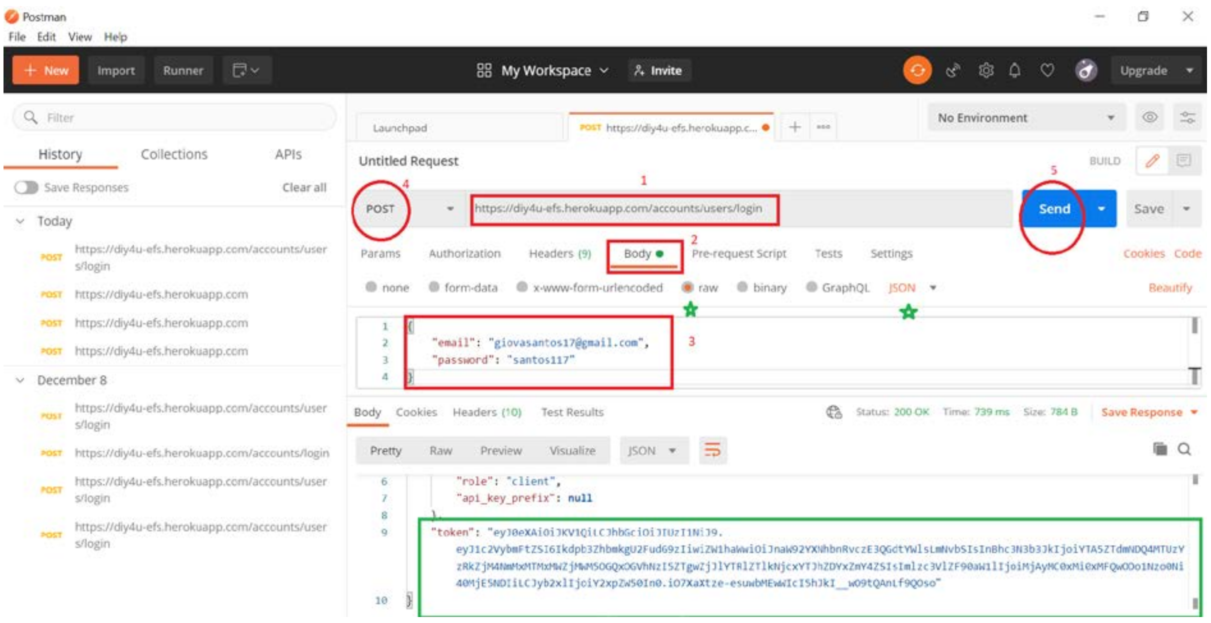
4. Authentication

As the API is private and secure, an authentication is required to give access to each resource. As The JSON Web Token (JWT) is one of the most common and robust authentication systems used by big companies, users can generate an access token and add it to the header of each request in the **"Authorization"** key. Note that by default, the generate token is validated for next 24h. Clients or users can get access tokens by login, as the API can be accessed by HTTP or HTTPs call, there are many methods to get a token. In order to demonstrate the authentication process, we start by presenting the Postman application method (for testing) and then the CURL command method which works with bash.
Note: Actually only Michael from P&G has an account and can access to our web-based interface.

4.1 Postman App

In order to get an authentication token using postman you should follow the steps.

1. Set the request URL to https://diy4u-efs.herokuapp.com/accounts/users/auth/login
2. Set the request method to POST
3. Define request body content type by selecting raw and JSON as data encoding.
4. Add user email and password in the body.
5. Finally, send the request by clicking in the send button and you should get in return a token if the user information are correct otherwise you get an error message for invalid credentials.


POST https://diy4u-efs.herokuapp.com/accounts/users/login

```
{
    "email": "giovasantos17@gmail.com",
    "password": "santos117"
}
```

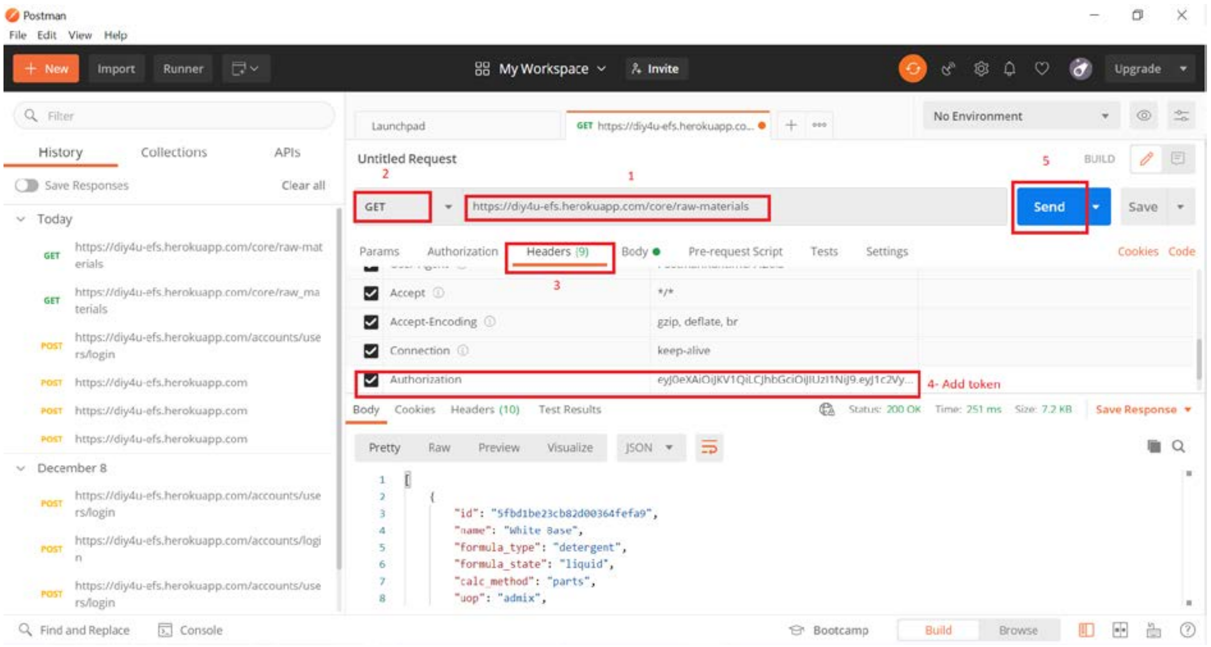
Status: 200 OK Time: 739 ms Size: 784 B

```
        "role": "client",
        "api_key_prefix": null
    },
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
        eyJ1c2VybmFtZSI6Ikdpb3ZhbmkgU2FudG9zIiwiZW1haWwiOiJnaW92YXNhbnRvczE3QGdtYWlsLmNvbSIsInBhc3N3b3JkIjoiYTA5ZTdmNDQ4MTUzY
        zRkZjM4NmMxMTMxMWZjMWM5OGQxOGVhNzI5ZTgwZjJlYTRlZTlkNjcxYTJhZDYxZmY4ZSIsImlzc3VlZF90aW1lIjoiMjAyMC0xMi0xMFQwODo1Nzo0Ni
        40MjE5NDIiLCJyb2xlIjoiY2xpZW50In0.iO7XaXtze-esuwbMEwWIcI5hJkI__wO9tQAnLf9QOso"
}
```

Figure 7: The authentification process

After generating a token, you need to add it in the request header, for that users have to follow these steps:

1. Select the headers tab
2. Add new Authorization key
3. Set the Authorization key value to your new generated token.
4. Do it for each request in order to authenticate your request. Note that if you did not change the tab postman will keep the same header, so not need to do it each time.


GET https://diy4u-efs.herokuapp.com/core/raw-materials

✓	Accept	*/*
✓	Accept-Encoding	gzip, deflate, br
✓	Connection	keep-alive
✓	Authorization	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2Vy...

Status: 200 OK Time: 251 ms Size: 7.2 KB

```
[
    {
        "id": "5fbd1be23cb82d00364fefa9",
        "name": "White Base",
        "formula_type": "detergent",
        "formula_state": "liquid",
        "calc_method": "parts",
        "uop": "admix",
```

Figure 8: Authorisation and interaction process

4.2. CURL command

In order to get an access token with the CURL method, users have to execute this command in a bash environment. Note that the **“Authorization”** key is not required in the login request.

```
curl -v -X POST https://diy4u-efs.herokuapp.com/accounts/users/auth/login
-H "Content-Type: application/json" \
-H "Authorization: Not matter" \
-d '{
  "email": "your.email@example.domain",
  "password": “My strong password”
}'
```

Figure 9: CURL Method

If the password and email match properly, users will receive their information and the access token in a JSON format.

5. Resources

5.1. Accounts

The account module is reserved to persons resources management. It contains three (03) main resources such organizations, customers, and users. These three resources are presented and detailed in the subsections below.

5.1.1. Customers

This collection contains information about each customer that is used in data analytics and later in the customization services to produce a better recommendation engine. The customer collection includes the following attributes in each document.

- **full_name**: the customer full name.
- **id (provided by the API)**: a unique identifier generated by the API to identify the customer.
- **external_id** : a unique identifier that identifies the customer in the organization system.
- **email (optional)**: the email of the customer
- **phone (optional)**: the phone number of the customer
- **owner_id (provided by the API):** used to identify each customer organization. This value is retrieved from the user authentication token during the customer creation, no need to manually add it.
- **descriptors**: contains extra information about the customer as plain objects. Accepted attributes are **gender**, **age**, **washer** and **profession**. Example {gender: "men", profession: “doctor”}
- **address**: contains information about customer addresses as plain objects. Accepted attributes are **country**, **state**, **city**, **zip_code**, **line**, **line_2**, **longitude**, **latitude.** Example {country: “France”}

5.1.2. Users

Users are the API clients and can use our modules to manage, and secure their resources database. Users can also be DIY4U stakeholders, to easily access our API resources. The user contains the following attributes in each document.

- **id (provided by the API)**: a unique identifier generated by the API to identify the user.
- **organization_id (Provided by the API):** the unique identifier of the organization to which the client belongs.
- **full_name**: the user full name.
- **email**: the email of the user. it should be unique.
- **role**: the role of the user. Possible value are “admin” or “client”
- **api_key (optional)**: a unique key generated for long time usage.
- **api_key_prefix(optional)**: a prefix to identified the API key

5.1.3. Organizations

The organizations provide a way to group users together in order to share the same resource. This feature allows the DIY4U stakeholders to have their own resources and give access to their selected clients. The organization must contain at least one user. All resources belong to the organization and not the client who created it. The organization contains the following attributes in each created document.

- **id (provided by the API)**: a unique identifier generated by the API to identify the user.
- **name**: the name of the organization. It should be unique.
- **email**: the email address to contact the organization.

5.2. Core

The core modules contain the main resource of our API. Core resources are basically all resources gathered and stored during the data collection task. These resources are used by machine learning for recommendation and data analytics resources. Resources such as raw materials, formulas, questions and adjustment rules are part of the core modules that are presented and detailed in the following subsections.

5.2.1. Raw materials

Raw materials are the base component used to create formulas. This resource allows us to maintain a single raw material and use it for all formulas that contain it. The raw material collection contains these following attributes in each document it contains.

- **id (provided by the API)**: a unique identifier generated by the API to identify the raw material.
- **name**: the name of the raw materials (not required to be unique but recommended)
- **formula_type (optional)**: define the type formula for which the raw material is used. The default value is **“detergent”**
- **formula_state**: define whether the raw material is used only for liquid, powder or both formulas. Possible value are **“liquid”**, **“powder”** and **“both”**
- **is_suspended (optional)**: this flag is used to inform whether the raw material is active or disable. The default value is false. If true the raw material is considered as deleted and will not be used for any operation.

- **calc_method**: define whether the raw material is used has parts of the formula or used to balance the formula. Possible value are **"parts"** and **"balance"**
- **uop:** define the type of component the raw material is. Possible values are **"admix"** and **"sprayon".**
- **owner_id:** used to identify each user raw materials. This value is retrieved from the user authentication token during the raw materials creation, no need to manually add it.
- **usage_tag (optional)**: defined which the properties of the formula the raw material affect. It is used during recommendation based on feedback. Possible values are **"other"**, **"scent"**, **"stain"**, **"color".**
- **unit_price:** define the raw material per unit.
- **co2_prod (optional):** define the carbon footprint generated during the production per unit.
- **co2_transport** (**optional**): define the carbon footprint generated during the transport per unit.

5.2.2. Formulas

The formula collection contains all validated and basic formulas used to create new products. The formula collection operates by maintaining a reference of all raw material needed for the creation. The formula collection contains the following attributes :

- **id (provided by the API)**: a unique identifier generated by the API to identify the formula
- **name**: the name of the formula (not required to be unique but recommended).
- **formula_type (optional)**: define the type products for which the formula is used. The default value is **"detergent"**
- **formula_state**: define whether the formula is used for liquid or powder formulas. Possible values are **"liquid"**, and **"powder"** .
- **is_suspended (optional)**: this flag is used to inform whether the formula is active or disable. The default value is false. If true the formula is considered as deleted and will not be used for any further operations.
- **owner_id:** used to identify each user's formula. This value is retrieved from the user authentication token during the formula creation, no need to manually add it.
- **is_basic (optional)**: define whether the formula is basic or not. the default value is false.
- **adjustustment_rules_id (optional)**: defined to determine the adjustment rules to be used during the recommendation for a given basic formula. This field is default to null and is only required for the basic formula.
- **constituents:** a list of objects that contains each raw material id and their weight in the formula. Example [{rms_id: weight}, {rms: weight} ...] . Note that the sum of all weights must be 1.

5.2.3. Questions

Define all possible questions that can be asked to the customer during the quiz. These questions will be used in the adjustment rules to define the adjustment level to be applied. The question collection contains these following attributes in each document it contains.

- **id (provided by the API)**: a unique identifier generated by the API to identify the question.
- **formula_type (optional)**: define the type formula for which the question is used. The default value is **"detergent"**

- **formula_state**: define whether the question is used only for liquid, powder or both formulas. Possible value are **“liquid”**, **“powder”** and **“both”**
- **title**: the title of the question (not required to be unique but recommended).
- **tag**: a short title that best describes the question.
- **description (optional)**: a text which best describes the question purpose and utilities.
- **owner_id:** used to identify each user question. This value is retrieved from the user authentication token during the question creation, no need to manually add it.
- **is_suspended (optional)**: this flag is used to inform whether the question is active or disable. The default value is false. If true the question is considered as deleted and will not be used for any operation.
- **choices**: a list of possible choices or answers. Each choice should have the following structure.
    - **choice_id (provided by the API)**: a unique identifier generated by the API to identify the question choice
    - **title**: the title of the question choice(not required to be unique but recommended).
    - **description**: a text which best describes the question choice purpose and utilities
    - **is_suspended (optional)**: this flag is used to inform whether the question choice is active or disable. The default value is false. If true the question choice is considered as deleted and will not be used for any operation.

5.2.4. Adjustment rules

Adjustment rules are used to define which percentage of each raw material should be kept in a given basic formula to find a theoretical formula that properly meets the customer requirements during the recommendation process. The adjustment rules collection contains the following attributes in each document.

- **id (provided by the API)**: a unique identifier generated by the API to identify the adjustment rules.
- **formula_type (optional)**: define the type formula for which the adjustment rules is used. The default value is **“detergent”**
- **formula_state**: define whether the adjustment rule is used for liquid or powder formulas. Possible value are **“liquid”**, **“powder”.**
- **title**: the title of the adjustment rule(not required to be unique but recommended).
- **owner_id:** used to identify each user adjustment rule. This value is retrieved from the user authentication token during the raw materials creation, no need to manually add it.

5.3. Analytics

The analytics module contains resources and analytic results generated during the client interactions with API. It includes resources such as consultations, orders and feedback. The module handles the management of the resource for each API organization. These resources are presented and detailed in the following subsections.

5.3.1. Consultations

Consultation contains information that is generated when customers take quizzes and request for recommended formulation based on their needs and expectations. The consultation resource provides a way to manage this kind of information and contain the following attributes in each document :

- **id (provided by the API)**: a unique identifier generated by the API to identify the consultation.
- **formula_type (optional)**: define the type of formula requested by the customer during the quiz. The default value is **“detergent”**
- **formula_state**: define whether customer requests for liquid or powder formula. Possible value are **“liquid”**, **“powder”**
- **owner_id (Provide by the API)**: The id of the organization in which the customer or the user belongs.
- **quiz_answers**: an object that contains the customers’ choice quiz. Example: {“question_id”: “choice_id” , ...}
- **customer_id**: The id of the customer who did the consultation.
- **create_at**: define when the consultations are occurred.
- **result (Provided by the API)**: an object that includes all data generated when the API is processing the request. The result information can be:
    - **theoretical_formula**: contains the theoretical formula obtained from the formula recommendation algorithm. Example: {“rms_id”: percentage, ...}
    - **basic_formula_id**: define the unique identifier of the basic formulation used to create the theoretical formula.
    - **recommendation**: contain the list of three sorted list of formulations to be recommended to the customer. Example: [{“formula_id”: value, “score”: value},]

5.3.2. Orders

After Quizzes, the API will recommend a formulation, and the costumer can confirm the order or not. If he confirms the order, then, the information related to the order is saved in the API in this organization to be used later for analytics purposes. Every order should refer to a unique consultation. The orders collection contains these following attributes :

- **id (provided by the API)**: a unique identifier generated by the API to identify the order.
- **customer_id**: The id of the customer who ordered.
- **owner_id (Provide by the API)**: The id of the organization in which the customer or the user belongs.
- **consultation_id**: the identifier of the consultation that was confirmed by the customer.
- **create_at(Provide by the API)**: the time when the customer confirms the order.
- **item**: an object that contains information about the ordered product. It contains attributes such as.
    - **item_id**: the identifier of the formulation chosen.
    - **quantity**: the quantity ordered in g.
    - **income**: the total income generated via the order.
    - **amount**: the total amount paid by the customer including shipment, tax, fees etc ….

5.3.3. Feedback

As the recommended formulation may not meet customer needs, RDIUP has defined and implemented a feedback system to gather customer feedback in order enhance the recommendation algorithm and also analyse this feedback to prioritise formulation and modify the weighted scores. The feedback collection comprises the following attributes:

- **id (provided by the API)**: a unique identifier generated by the API to identify the feedback.
- **customer_id**: The id of the customer who gives the feedback.
- **order_id**: the identifier of the order that is the origin of the feedback.
- **stain_evaluation**: define how good was the stain removal capacity. Accepted values are (“very-weak”, “weak”, “perfect”, “strong”, “very-strong”)
- **scent_evaluation**: define the level of the scent intensity. Accepted values are (“very-weak”, “weak”, “perfect”, “strong”, “very-strong”)
- **satisfied**: determine whether the customer is satisfied or not. Accepted values are (“true”, “false”)
- **comment**: a text that describes the customer feelings and experiences.

6. Management of resources

Managing resources allows to make HTTP or HTTPs requests to the API in order to make change, get data or add new data to the resources. The API contains two main modules, the core that contains primary resources, accounts that contains user account related to resources and analytics which contains resources related to the features and analytic results. Each resource URL contains the module name before the resource name.

6.1. Manage Accounts resources

The account module contains resources that are prefixed by accounts keyword. All resources available on the accounts module are **users**, and **customers.** To manage each resource, clients need to replace the {resource-name} by the corresponding name and hosted by https://diy4u-efs.herokuapp.com/. Note that the body of each request should contain validated data for the given resource, all attributes are described in the resource sections.

Request Type	Request URL	Body	Action	Return Data
POST	host/accounts/organization	organization name and email plus main user data	Create organization with a given user	Return user and auth token
POST	host/accounts/users/auth/register	user object in JSON	add new users to the organization	The new user
POST	host/accounts/users/auth/login	email and password	login user	User information and a token
GET	host/accounts/{reso	-	Retrieve all item of	All item of a

	urce-name}		a given resource	given resource (only for the user)
GET	host/accounts/{resource-name}/id	-	Retrieve an item of a given resource by it id	An item of a given resource
POST (only for customer)	host/accounts/{resource-name}	resource object in JSON	create new resource item	The new created resource
PUT	host/accounts/{resource-name}	resource object in JSON	Update item of the resource	Error if failed or 204 status code if success
DELETE	host/accounts/{resource-name}/id	-	Delete an item of the resource by the item id	Error if failed and 204 status

6.2. Manage core resources

The core module contains many resources that are prefixed by the core keyword. All resources available on the core module are **raw-materials**, **formulas**, **questions**, **adjustment-rules.** To manage each resource, users have to replace the {resource-name} by the corresponding name hosted by https://diy4u-efs.herokuapp.com/ . Note that the body of each request should contain validated data for the given resource, all attributes are described in the resource sections.

Request Method	**Request URL**	**Request Body**	**Return Data**	**Action**
GET	host/core/{resource-name}	-	All item of a given resource (only for the user)	Retrieve all item of a given resource
GET	host/core/{resource-name}/id	-	An item of a given resource	Retrieve an item of a given resource by it id
POST	host/core/{resource-name}	resource object in JSON	The new created resource	Create new resource item
PUT	host/core/{resource-name}	resource object in JSON	Error if failed or 204 status code if success	Update item of the resource
DELETE	host/core/{resource-name}/id	-	Error if failed and 204 status code if success	Delete an item of the resource by the item id

6.3. Manage analytics resource

The analytics module contains many resources that are prefixed by the analytics keyword. All resources available on the core module are **consultations**, **orders**, **feedback.** To manage each resource replace the {resource-name} by the corresponding name and hosted by https://diy4u-efs.herokuapp.com/ .

Request Method	**Request URL**	**Request Body**	**Return Data**	**Action**
GET	host/analytics/{resource -name}	-	All item of a given resource (only for the user)	Retrieve all item of a given resource
GET	host/analytics/{resource -name}/id	-	An item of a given resource	Retrieve an item of a given resource by it id
POST	host/analytics/{resource -name}	resource object in JSON	The new created resource	Create new resource item
PUT	host/analytics/{resource -name}	resource object in JSON	Error if failed or 204 status code if success	Update item of the resource
DELETE	host/analytics/{resource -name}/id	-	Error if failed and 204 status code if success	Delete an item of the resource by the item id

7. Conclusion

The DIY4U EFs functionalities develops by DIY4U was built based on the well REST API that facilitate the communication with the modules of the DIY4U platform. This document describes how to interact with the developed API. As the API is protected and secure, an authentication JSON Web token is required to get access, two methods of access token generation were defined. All resources of the API were described in detail and the way to manage them was also presented. The user interface defined by RDIUP has allowed to demonstrate and test these services, features and Functions. For the future demonstrations in real condition, our API has to be hosted in a performant cloud server and the database will be extended for multi-users.