The Africa PID Alliance Initiative

THE AFRICA PID ALLIANCE INITIATIVE

DIGITAL OBJECT CONTAINER IDENTIFIER (DOCiD$_{TM}$)

Phase 1

The Africa PID Alliance Initiative

This is the first draft of the Africa PID Alliance Digital Object Identifier (DOCiD) documentation

2024

TABLE OF CONTENTS

ABBREVIATIONS AND ACRONYMS

API - Application Programming Interface

CRUD - Create, Read, Update, Delete

DOCID - Document Identifier

RDBMS - Relational Database Management System

SQL - Structured Query Language

SSL - Secure Sockets Layer

TLS - Transport Layer Security

URL - Uniform Resource Locator

DEFINITION TERMS

- **Docker:** Imagine Docker as a set of magic boxes that can hold everything your app needs to run - like the software it uses, its settings, and even the computer it runs on. It keeps everything neatly packed so that you can easily move your app from one place to another without any mess.
- **MicroServices:** Think of microservices like a team of specialized workers in a factory. Each worker (microservice) has a specific job, and they work together

efficiently to build the final product (your app). This approach makes it easier to manage and update different parts of your app independently.

- **Flask:** Picture Flask as a chef who knows how to cook up web applications. It's a Python-based framework that helps you create web apps by handling things like serving web pages, processing user inputs, and interacting with databases.

- **React:** Imagine React as a super-smart interior designer for your web pages. It's a JavaScript library that helps you build interactive and dynamic user interfaces (UIs) for your web apps. React makes creating components like buttons, forms, and menus that respond to user actions easy.

- **Frontend:** The front end of a web app is like the shop window that customers see. It's the part of the app that users interact with directly, such as clicking buttons, filling out forms, and viewing content like text and images.

- **Backend:** The backend of a web app is like the hidden machinery behind the scenes. It handles tasks like processing data, interacting with databases, and sending information to the front end. Users don't see the backend directly but rely on it to make the app work smoothly.

- **Host:** Think of a host as a friendly hotel that welcomes guests (your app). It provides a comfortable space (server or computer) where your app can stay and be accessible to users over the internet.

- **Container:** Picture a container as a portable box that holds everything your app needs to run, like software, files, and settings. Containers are like mini virtual computers that can be easily moved, duplicated, and managed.

- **Python:** Python is a versatile tool that can be used for many tasks, from building web apps (using frameworks like Flask) to data analysis, artificial intelligence, and more. It's a popular programming language known for its readability and ease of use.

- **JavaScript:** JavaScript is like the secret sauce of the web. It's a programming language that adds interactivity and dynamic features to web pages. With JavaScript, you can create animations, validate forms, fetch server data, and make web apps feel more alive.

INTRODUCTION

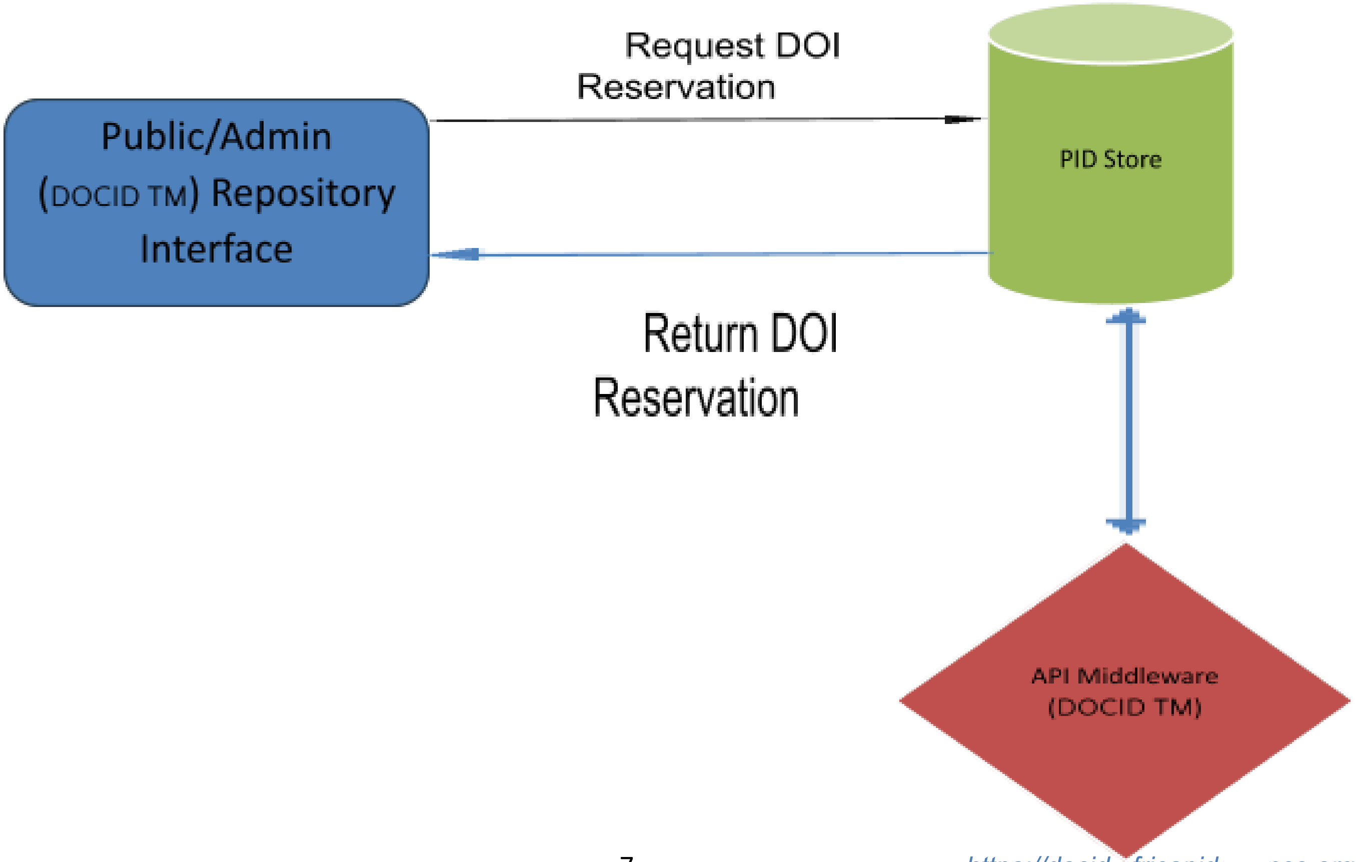
The Africa PID Alliance is undertaking a transformative mission at the convergence of technological progress and cultural preservation to address the urgent need for robust data infrastructures in Africa. With a focus on fostering innovation in indigenous knowledge and cultural heritage, particularly in the realm of patents and digital objects, the Alliance recognizes the financial constraints faced by some universities and the prevalence of gray literature in Africa. To address this, a multilinear digital object identification system is proposed, seamlessly linking local handles and DOIs to manage diverse research content effectively.

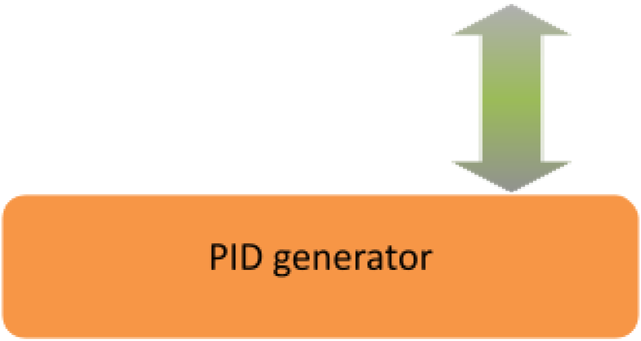
The multilinear data model extends to indigenous knowledge, uniting biocultural attributes and scientific data within a single digital object container (DOCiD™), fostering a comprehensive understanding. Leveraging CORDRA and DOCiD App, our digital infrastructure integrates their strengths in digital object repository management and library presentation, ensuring a solid foundation for the multilinear data model. The integration layer acts as a bridge, facilitating communication between CORDRA and DOCiD™ App. Our architecture prioritizes access and retrieval through DOCiD App, emphasizing user-friendly interfaces and stringent error handling. The commitment to innovation is showcased in the implementation of a multilinear DOCiD™ Infrastructure Pilot, utilizing open-source software for automatic identifier creation and management of digital objects within the APA Initiative framework. As we embark on this journey, the Africa PID Alliance invites you to be a signatory to an open letter, expressing support for the advancement of digital infrastructure in Africa and the preservation of indigenous knowledge and cultural heritage.

MAJOR COMPONENTS

The DOCiD Ecosystem comprises a PID Generator and resolution component, Object repository and an integration middleware responsible for 2-way integration of the two systems (PID Generator and Object repository)

The illustration below summarizes the interlinking of the 3 major components:


Request DOI Reservation
Public/Admin (DOCID TM) Repository Interface
PID Store
Return DOI Reservation
API Middleware (DOCID TM)



1. PID Reservation & Generator Component

The PID generator component of the DOCiD eco-system functions to achieve the following:

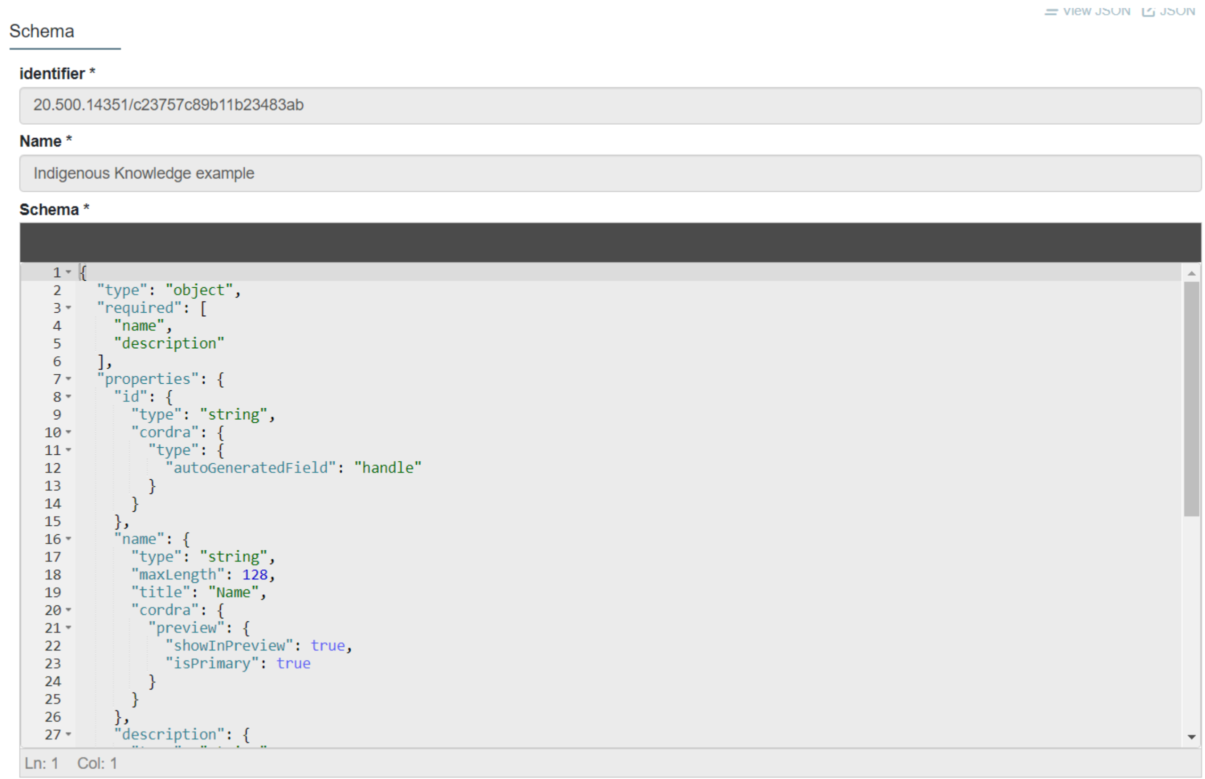
1. PID generation
2. PID resolution
3. Object address resolution
4. Object types Definition

Customized around Cordra, this component is a highly configurable application for managing digital objects with resolvable identifiers at scale while providing the following features

1. Configurability: Schemas, business rules, access controls, and much more can be configured using settings and lifecycle hooks, all using the built-in admin interface.
2. Scalability: Enables horizontal scaling and concurrency techniques to support large-scale demand. Provides a load-sharing front to backend services.

Object Types definition:

Indigenous Knowledge example



Schema

identifier *

20.500.14351/c23757c89b11b23483ab

Name *

Indigenous Knowledge example

Schema *

```
{
  "type": "object",
  "required": [
    "name",
    "description"
  ],
  "properties": {
    "id": {
      "type": "string",
      "cordra": {
        "type": {
          "autoGeneratedField": "handle"
        }
      }
    },
    "name": {
      "type": "string",
      "maxLength": 128,
      "title": "Name",
      "cordra": {
        "preview": {
          "showInPreview": true,
          "isPrimary": true
        }
      }
    },
    "description": {
```

2. Digital Object Repository

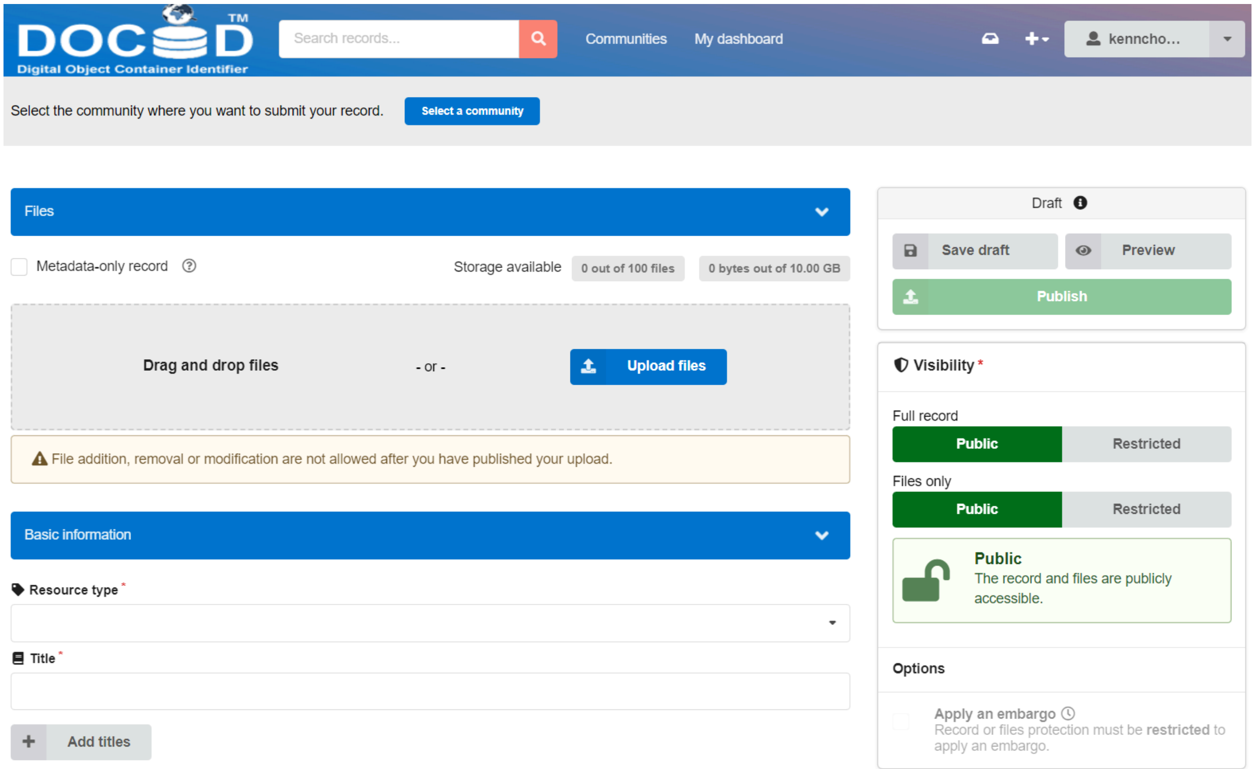
This component of the DOCiD eco-system functions to serve as object repository where objects with persistent identifiers (DOIs) are defined, located by way of search and regularly updated by owners of the objects. Customized around Invenio software, this application setup has the following components:

1.	Database Back-end	PostgreSQL
2.	Search	Elasticsearch
3.	Application	Developed in Python, Flask Framework

The digital objects defined in this module have their DOIs / PIDs generated from the PID generator component through the API middleware component.

Object Definition:

V1 of Add Publication Modal



Select the community where you want to submit your record. Select a community

□ Metadata-only record

Storage available 0 out of 100 files 0 bytes out of 10.00 GB

Drag and drop files - or - Upload files

⚠ File addition, removal or modification are not allowed after you have published your upload.

Resource type *

Title *

Visibility *

Full record: [✓ Public] Restricted

Files only: [✓ Public] Restricted

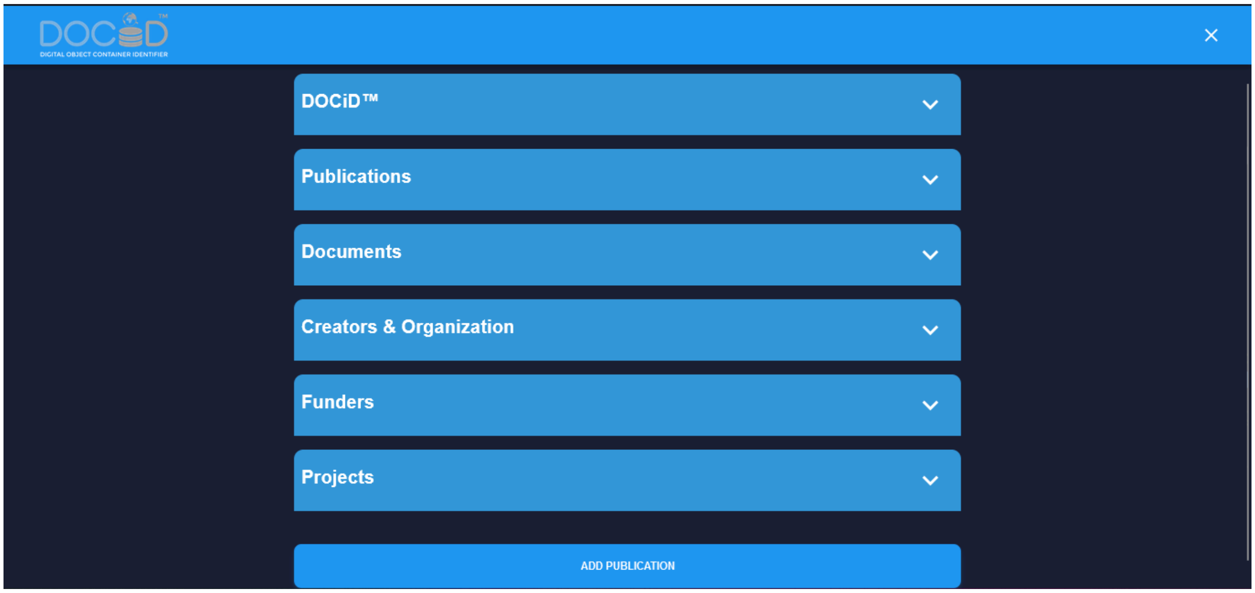
Public
The record and files are publicly accessible.

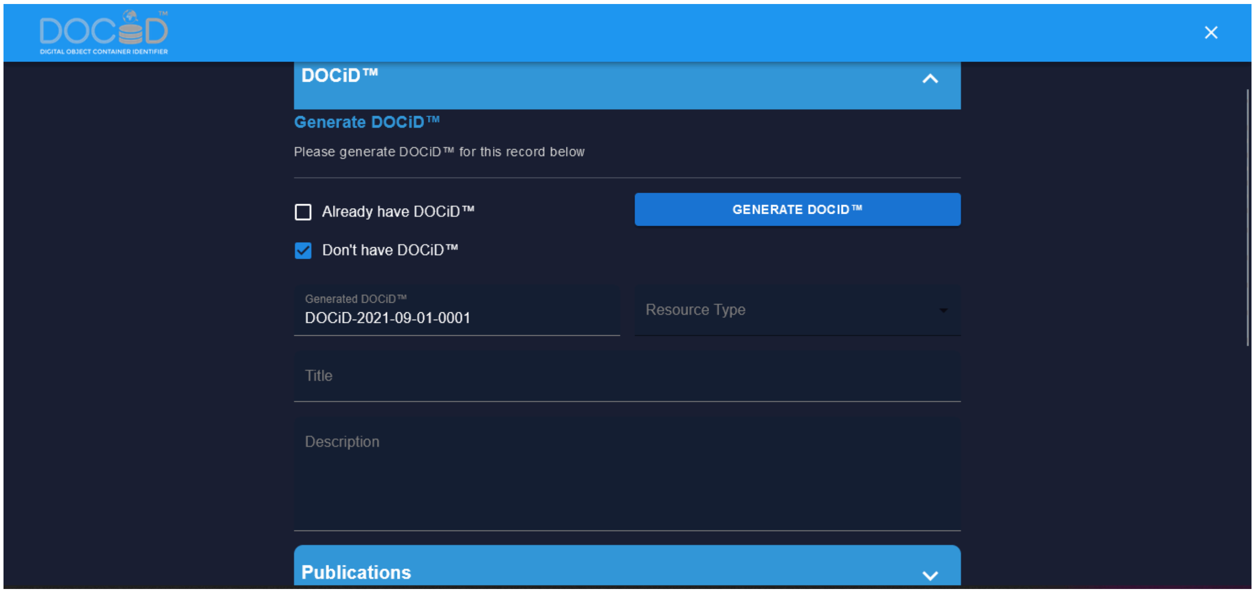
Options

□ Apply an embargo
Record or files protection must be **restricted** to apply an embargo.

This application is extensible and customizable to included objects desired fields that can then be configured for search capabilities

V2 of Add Publication Modal





Generate DOCiD™

Please generate DOCiD™ for this record below

□ Already have DOCiD™

☑ Don't have DOCiD™

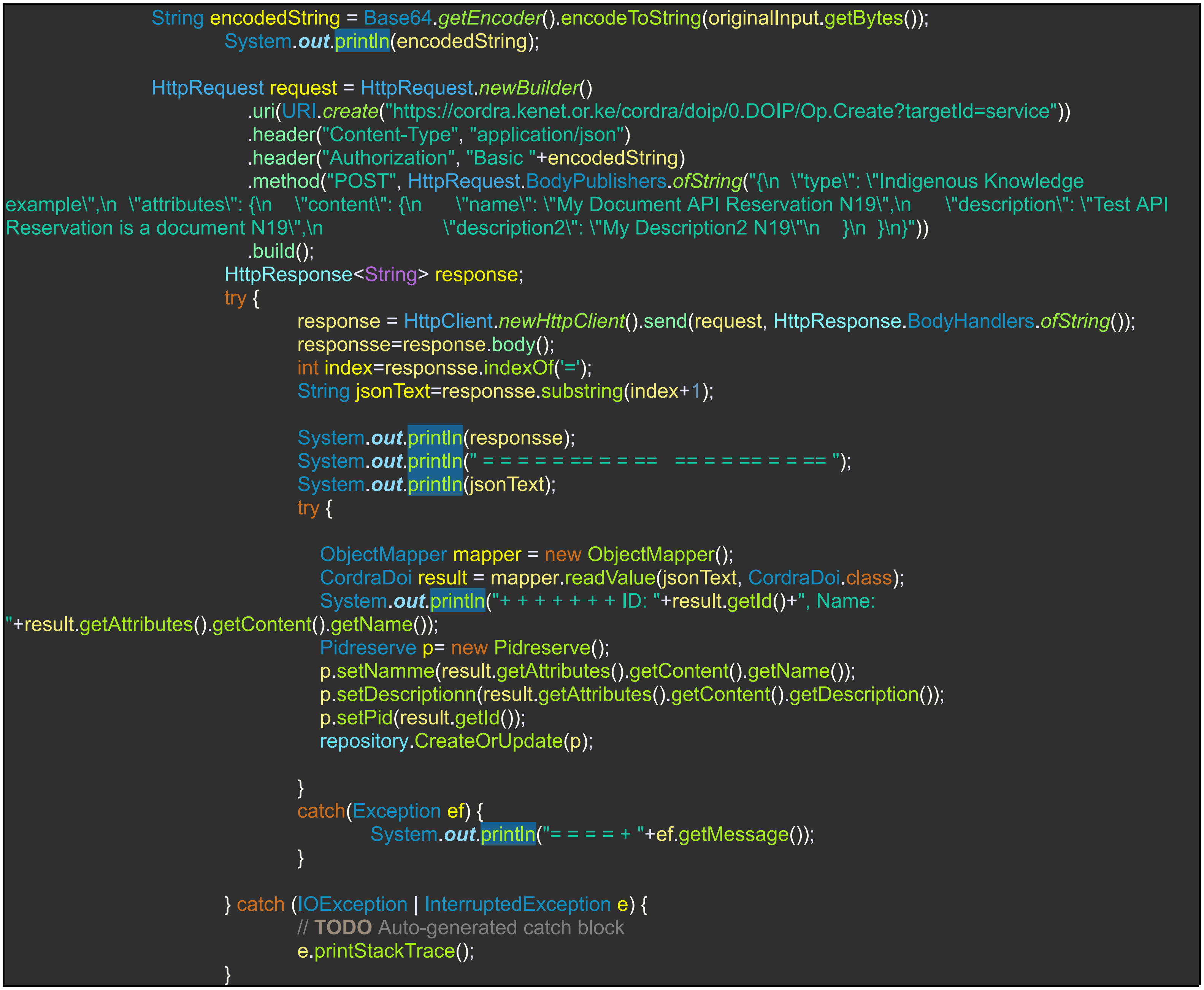
Generated DOCiD™
DOCiD-2021-09-01-0001

3. Middleware API Component

This component of the DOCiD ecosystem integrates the PID generator with the Digital object repository system. The component upon invoking instructs PID generator to create a new PID and serves the generated PID to the Digital object repository solution which utilizes it to register a newly created object.

Sample code:

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.util.Base64;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.example.demo.model.CordraDoi;
import com.example.demo.model.PidReserveDb;
import com.example.demo.model.Pidreserve;
import com.fasterxml.jackson.databind.ObjectMapper;

@RestController
@RequestMapping("/invenio/middleware")
public class CordraInvenioMW2 {
        private final PidReserveDb repository;
        public CordraInvenioMW2(PidReserveDb db) {
                this.repository=db;
        }
        @GetMapping
  public String getReservation() {

                String responsse="No Data Found";

                String originalInput = "admin:password";
```



```
                String encodedString = Base64.getEncoder().encodeToString(originalInput.getBytes());
                        System.out.println(encodedString);

                HttpRequest request = HttpRequest.newBuilder()
                                .uri(URI.create("https://cordra.kenet.or.ke/cordra/doip/0.DOIP/Op.Create?targetId=service"))
                                .header("Content-Type", "application/json")
                                .header("Authorization", "Basic "+encodedString)
                                .method("POST", HttpRequest.BodyPublishers.ofString("{\n  \"type\": \"Indigenous Knowledge
example\",\n  \"attributes\": {\n    \"content\": {\n      \"name\": \"My Document API Reservation N19\",\n      \"description\": \"Test API
Reservation is a document N19\",\n                    \"description2\": \"My Description2 N19\"\n    }\n  }\n}"))
                                .build();
                        HttpResponse<String> response;
                        try {
                                response = HttpClient.newHttpClient().send(request, HttpResponse.BodyHandlers.ofString());
                                responsse=response.body();
                                int index=responsse.indexOf('=');
                                String jsonText=responsse.substring(index+1);

                                System.out.println(responsse);
                                System.out.println(" = = = = = == = = ==   == = = == = = == ");
                                System.out.println(jsonText);
                                try {

                                    ObjectMapper mapper = new ObjectMapper();
                                    CordraDoi result = mapper.readValue(jsonText, CordraDoi.class);
                                    System.out.println("+ + + + + + + ID: "+result.getId()+", Name:
"+result.getAttributes().getContent().getName());
                                    Pidreserve p= new Pidreserve();
                                    p.setNamme(result.getAttributes().getContent().getName());
                                    p.setDescriptionn(result.getAttributes().getContent().getDescription());
                                    p.setPid(result.getId());
                                    repository.CreateOrUpdate(p);

                                }
                                catch(Exception ef) {
                                        System.out.println("= = = = + "+ef.getMessage());
                                }

                        } catch (IOException | InterruptedException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
```

```
        return responsse;
    }
}
```

PIDs Log sample

Field	Description
ID	INT, Auto
Name	Varchar
Description	Varchar
PID	Varchar
PID_Reserved	Bit
PID_ReservedDate	DatetTime
PID_Assigned	Bit
PID_AssignedDate	DateTime
PID_ReservedBy	Varchar
PID_AssignedBy	Varchar
ObjectRef	Varchar - DOCID object ref

PROJECT STRUCTURE

The DOCID Integration App project contains several folders/directories and files that are useful to the operation and functionality of the app. It uses a Microservices architecture approach where four core services are being utilized and include:

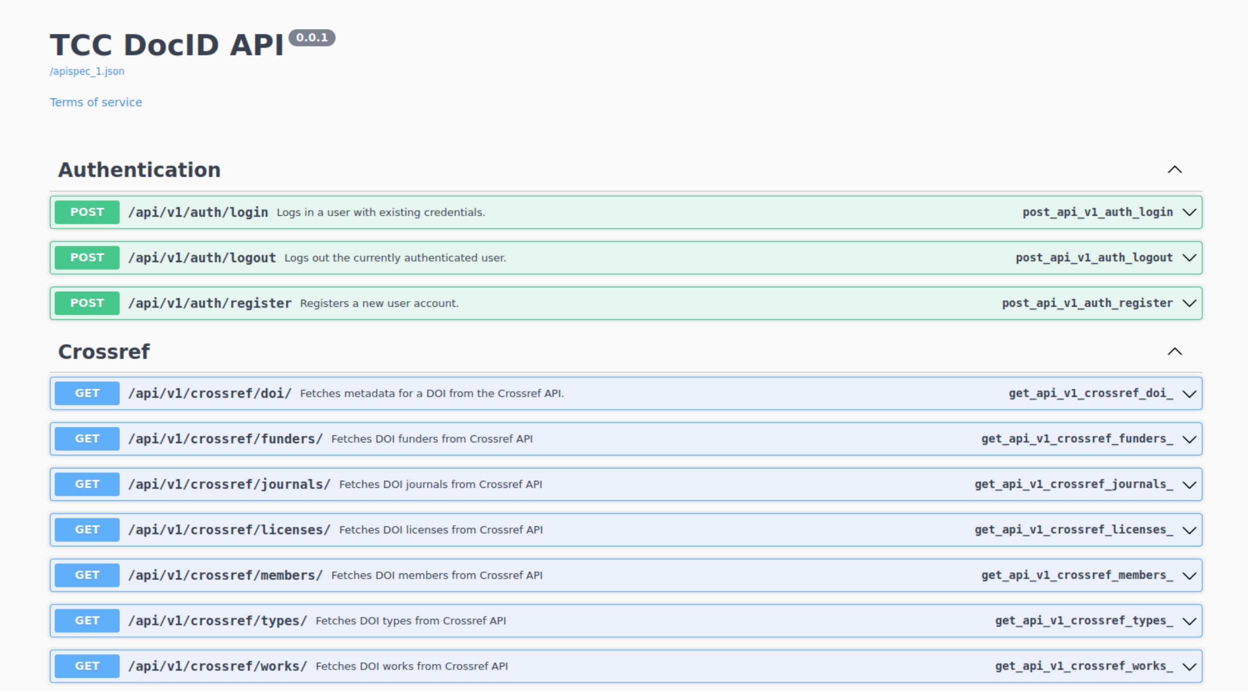
Frontend Service:

This service consists of the user interface that the users, e.g. librarians, will interact with. Consisting of input forms, dropdowns, buttons, etc. Basically, this is what the end user sees when they open the application through the designated URL. The front end has been developed using React, a popular JavaScript library that is used to come up with interactive and intuitive user interfaces. Other software and dependencies used include *react-boostrap, Material-UI, Tailwind CSS, Material-Tailwind, Axios and react-router-dom*. The end product is a front-end web application.

Backend Service

This service consists of functionality and logic that links the front-end service to the database and the web server. It has been written in Flask, a Python microframework used to develop web applications. The reason behind choosing Flask is that it is highly scalable, meaning that it can be designed to handle small operations to very large operations, and also has multiple extensions and plugins that help design a custom solution like DOCID App. Additional libraries that have been used include *psycopg2, python-dotenv, flask-cors, gunicorn, habanero,* and *flask-sqlalchemy*. The end product is a web application that provides a rich Application Programming Interface (API) that supports user authorization and authentication, publishing files and records, and retrieval of the published content.

Below are the backend APIs for DocID to pull data from backend or from external APIs like ROR,RAiD, ORCID,CrossRef, Datacite and DOCiD publications.



TCC DocID API 0.0.1

/apispec_1.json

Terms of service

Authentication

POST /api/v1/auth/login Logs in a user with existing credentials. post_api_v1_auth_login

POST /api/v1/auth/logout Logs out the currently authenticated user. post_api_v1_auth_logout

POST /api/v1/auth/register Registers a new user account. post_api_v1_auth_register

Crossref

GET /api/v1/crossref/doi/ Fetches metadata for a DOI from the Crossref API. get_api_v1_crossref_doi_

GET /api/v1/crossref/funders/ Fetches DOI funders from Crossref API get_api_v1_crossref_funders_

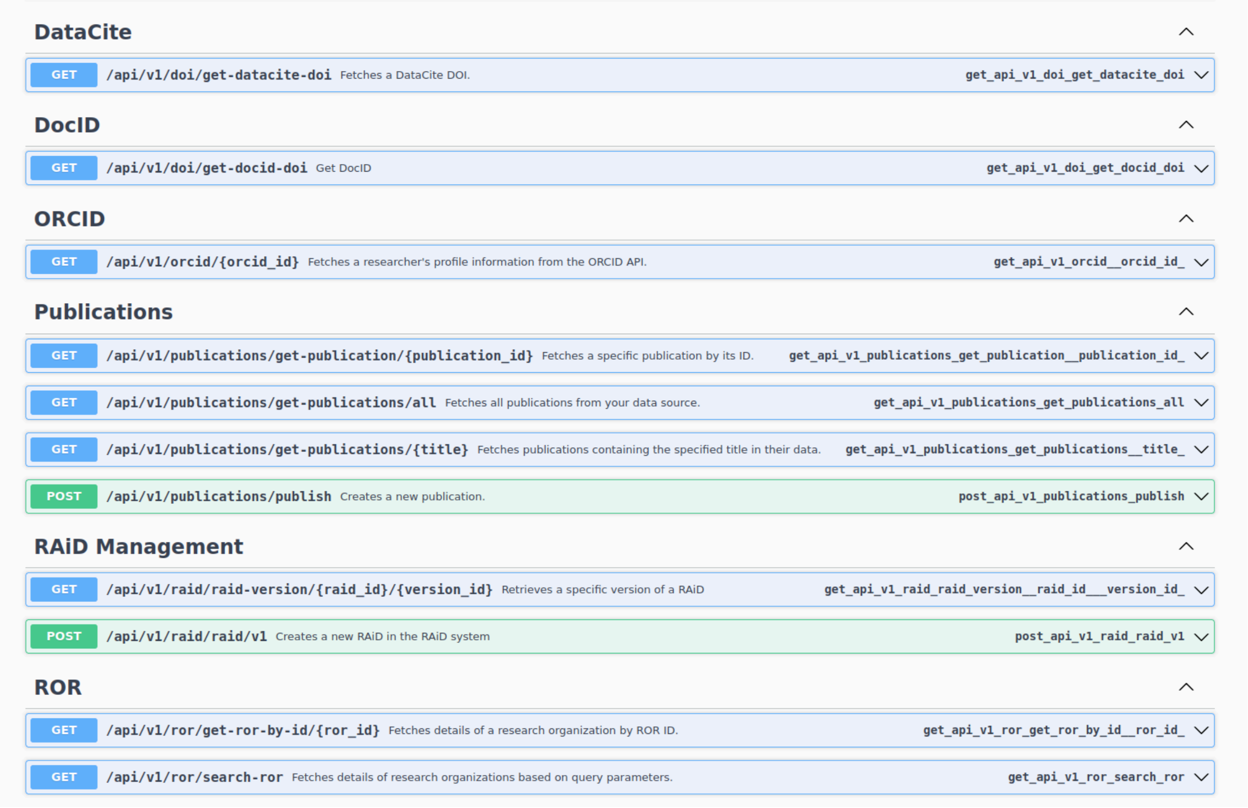
GET /api/v1/crossref/journals/ Fetches DOI journals from Crossref API get_api_v1_crossref_journals_

GET /api/v1/crossref/licenses/ Fetches DOI licenses from Crossref API get_api_v1_crossref_licenses_

GET /api/v1/crossref/members/ Fetches DOI members from Crossref API get_api_v1_crossref_members_

GET /api/v1/crossref/types/ Fetches DOI types from Crossref API get_api_v1_crossref_types_

GET /api/v1/crossref/works/ Fetches DOI works from Crossref API get_api_v1_crossref_works_



DataCite

GET /api/v1/doi/get-datacite-doi Fetches a DataCite DOI. get_api_v1_doi_get_datacite_doi

DocID

GET /api/v1/doi/get-docid-doi Get DocID get_api_v1_doi_get_docid_doi

ORCID

GET /api/v1/orcid/{orcid_id} Fetches a researcher's profile information from the ORCID API. get_api_v1_orcid__orcid_id_

Publications

GET /api/v1/publications/get-publication/{publication_id} Fetches a specific publication by its ID. get_api_v1_publications_get_publication__publication_id_

GET /api/v1/publications/get-publications/all Fetches all publications from your data source. get_api_v1_publications_get_publications_all

GET /api/v1/publications/get-publications/{title} Fetches publications containing the specified title in their data. get_api_v1_publications_get_publications__title_

POST /api/v1/publications/publish Creates a new publication. post_api_v1_publications_publish

RAiD Management

GET /api/v1/raid/raid-version/{raid_id}/{version_id} Retrieves a specific version of a RAiD get_api_v1_raid_raid_version__raid_id___version_id_

POST /api/v1/raid/raid/v1 Creates a new RAiD in the RAiD system post_api_v1_raid_raid_v1

ROR

GET /api/v1/ror/get-ror-by-id/{ror_id} Fetches details of a research organization by ROR ID. get_api_v1_ror_get_ror_by_id__ror_id_

GET /api/v1/ror/search-ror Fetches details of research organizations based on query parameters. get_api_v1_ror_search_ror

Database Service

This service consists of a Relational Database Management System (RDBMS) which in this case is using PostgreSQL, an open-source, highly scalable, and enterprise-level software. The work of PostgreSQL is to support Create, Read, Update, Delete (CRUD) operations in the DOCID Integration App.

Nginx Service.

This service consists of a web server, Nginx, a popular open-source web server, and a reverse proxy server. It's designed to efficiently handle a large number of concurrent connections and to serve static content quickly. It serves and provides functionalities such as load balancing, SSL/TLS termination, and caching which ensure communication and security between the frontend and the backend services.

DEPLOYMENT STEPS

Front-End

1. In a clean operating system, install Git, which is optional but necessary. Git is a tool that is used for versioning and collaboration in developing software between developers and other teams.
2. Install Firebase which is optional. The Firebase Realtime Database is a cloud-hosted NoSQL database that lets organizations store and sync data in real time across all of their users' devices. This makes it easy to build apps that are always up to date, even when users are offline.
3. Through the terminal Log into firebase under the project directory by running “firebase login” and then ‘firebase init’ after successful logging in.

4. Select from the option of existing ones if you have a project already under github pushing optional.
5. Generate a build folder by writing ‘build’ when asked about public folder to be used
6. All the options should be accepted by ‘yes’ apart from the github pushing option by reply to ‘No’
7. After that run ‘npm run build’ to build the project content for production.
8. after a successfully build then run ‘firebase deploy’

DESIRED HOSTING SPECS ON EACH COMPONENT (PID GENERATOR, AND OBJECT REPO)

The Digital Object Repository and the PID Generator current servers have 8GB of memory, a 4-core CPU with a clock speed of 2095.078 MHz, and a storage capacity of 60GB. Below are the desired server specifications.

PID Generator			Quantity
1	Server	Cloud Servers	4 (2 active, 2 Failover)

2	Server CPU	16 VCPUs	x2 servers
3	RAM	32GB	x2 servers
4	Storage	5 TB	x2 servers
5	Load Balancer	Software load balancer	1
6	Backup Storage	3TB - Cloud block storage	1