Open Data Management & Cloud

Thomas Axel Deponte

November 11, 2024

Enhancing Open Source Product Development through Product Lifecycle Management

1 Introduction

This document presents an open data management and cloud examination project that focuses on addressing the challenges faced by companies or communities developing open-source products in managing their product lifecycle effectively. The project aims to design a simple but effective platform lifecycle management (PLM) solution that adheres to the FAIR (**Findable**, **Accessible**, **Interoperable**, **Reusable**) guidelines for data management and cloud deployment strategies.

The project is divided into three main parts:

1. **Model Design**: This section outlines the conceptual framework of the PLM solution, including its objectives, scope, and key features.

2. **Implementation Prototype**: In this part, a prototype web application is developed using Django, focusing solely on the API. The prototype serves as a proof-of-concept to demonstrate the feasibility and functionality of the PLM solution.

3. **Cloud Solutions Evaluation**: This section evaluates various cloud solutions for deploying the PLM solution. The evaluation considers factors such as scalability, security, cost, and compatibility with open-source technologies. Additionally, a private deployment strategy is explored in case of a company to ensure data sovereignty and compliance with internal policies.

Throughout this project, the focus will be on creating an efficient, scalable, and secure PLM solution that enhances collaboration, transparency, and sustainability in open-source product development communities.

2 Preliminary Data Analysis

A servo drive controller, for instance, is a complex product comprised of various physical components and assemblies that require meticulous tracking. Typically, developers maintain a comprehensive list of these parts in a table known as the Bill of Materials (BOM). For each part, it's beneficial to know which manufacturer can produce or sell it, their unique code for identification, whether it's obsolete, and its functional limitations (as outlined in the datasheet).

There may be several alternatives available for each part. Additionally, developers need to keep a record of production documents such as CAD files, Gerber files, etc. It's important to note that these documents can undergo revisions, just like the parts themselves.

Each part can be associated with a link to the design source repository, which may include code, CAD models, schematics, simulations, and more. A part can be released, meaning it cannot accept any new modifications, and a new part name must be created in such cases.

It's beneficial to track the modification date of all this information for accurate record-keeping and efficient management.

3 Internal part number

One key aspect of PLM is the use of Internal Part Numbers (IPNs), which serve as unique identifiers for all components, software, and documentation within an organization. In this context, I have chosen to implement a semi-structured IPN format such as 'HW-RES003-v01r02'. This decision is based on the benefits that this format offers in terms of clarity, simplicity, and compatibility with existing databases.

The initial part of the IPN, 'HW', is an arbitrary identifier unique to our organization, which helps avoid collisions with common IPNs already in use in other databases. This ensures that each component can be accurately and uniquely identified within our system. The following 'RES' denotes the category of the part, in this case, a resistor. This categorization allows for easy sorting and retrieval of parts during the product development lifecycle.

The remaining sections of the IPN, '003', 'v01', and 'r02', represent the variant and revision of the part. The use of a sequential number ('003') ensures that each new part within the same category has a unique identifier. This flexibility is crucial for managing variations and updates to components throughout their lifecycle.

The inclusion of 'v01' and 'r02' allows us to track changes and versions of the part. The 'v01' indicates the first version of the resistor, while 'r02' signifies the second revision. This level of detail is essential for maintaining accurate records and ensuring that the correct version of a component is used in production.

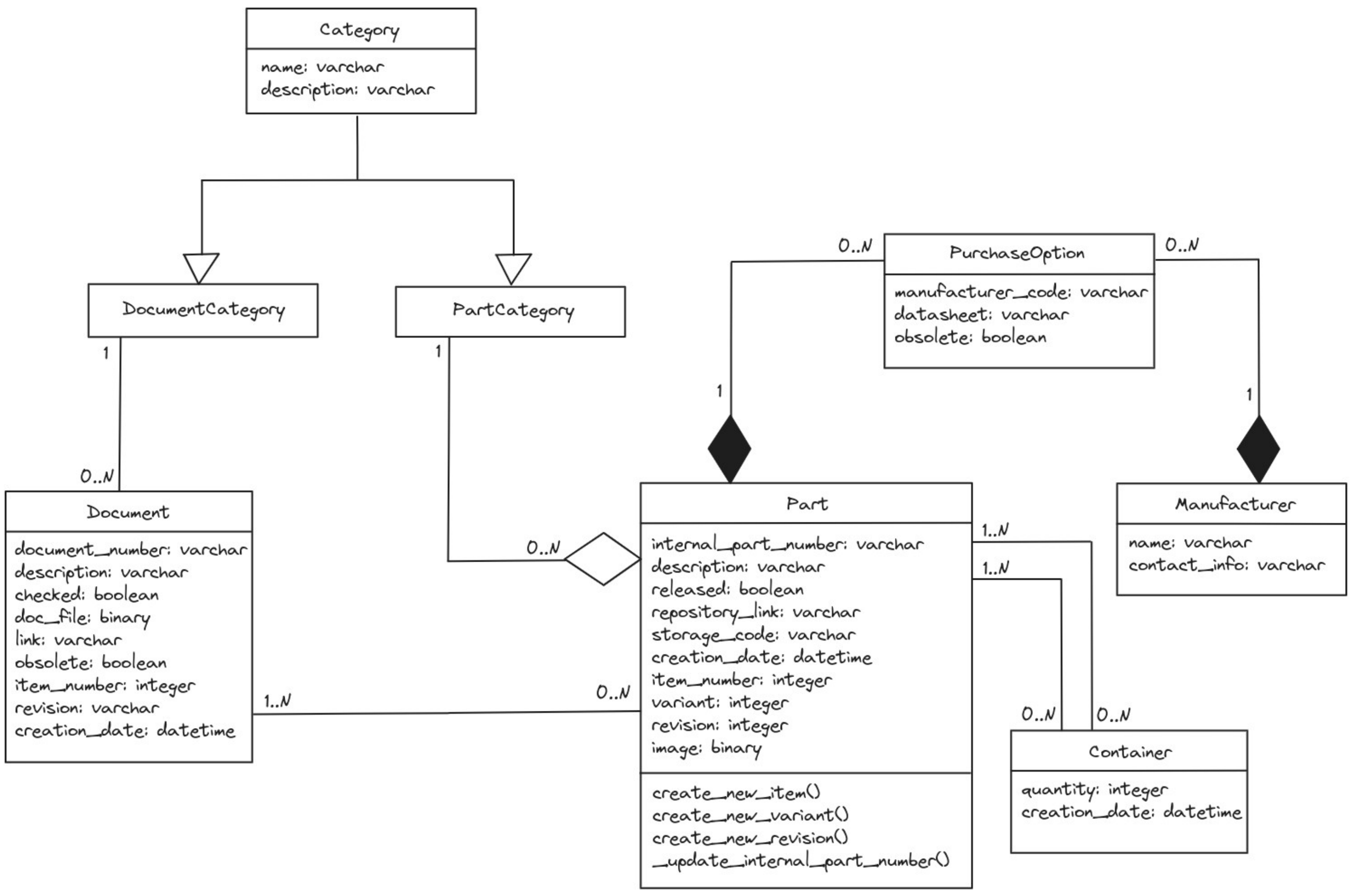
Is important to note that the variant and revision are attributes that give (temporal) information related only with the part itself.

More information can be found in the links below.

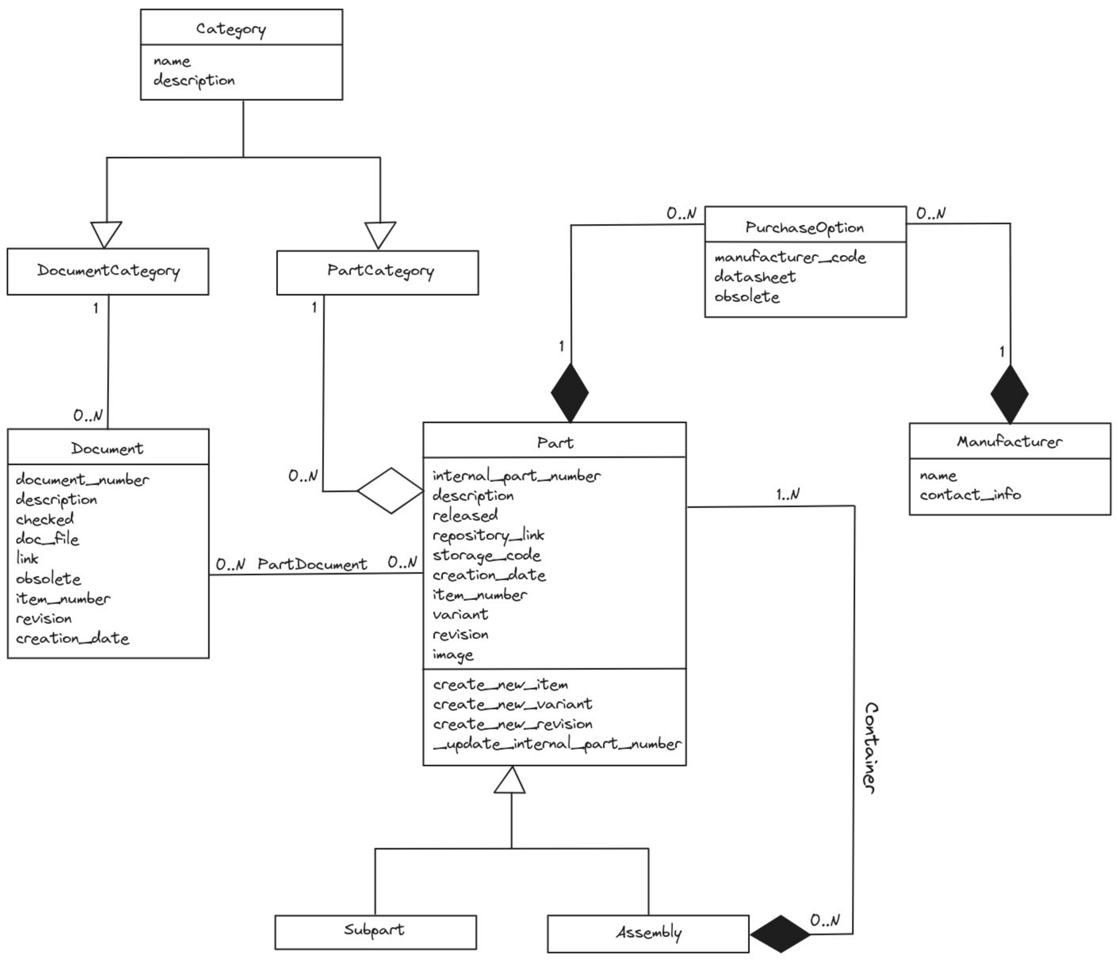
`https://github.com/git-plm/parts/blob/main/partnumbers.md`
`https://en.wikipedia.org/wiki/Part_number`

4 Model Design

UML Diagram



Restructured Model Diagram with Attributes domains



I eliminated the entities 'Subpart' and 'Assembly' from Part specializations as they didn't have any unique attributes or operators differentiating them. To address this, I created a new table that includes the attributes quantity and creation date to represent an item in a Bill of Materials (BOM).

I maintained the Category Generalization because I plan to implement it using Django ORM with an abstract class. For optimal query efficiency and ease of development, I chose to use single surrogate keys over natural or composite keys.

5 Data model implementation

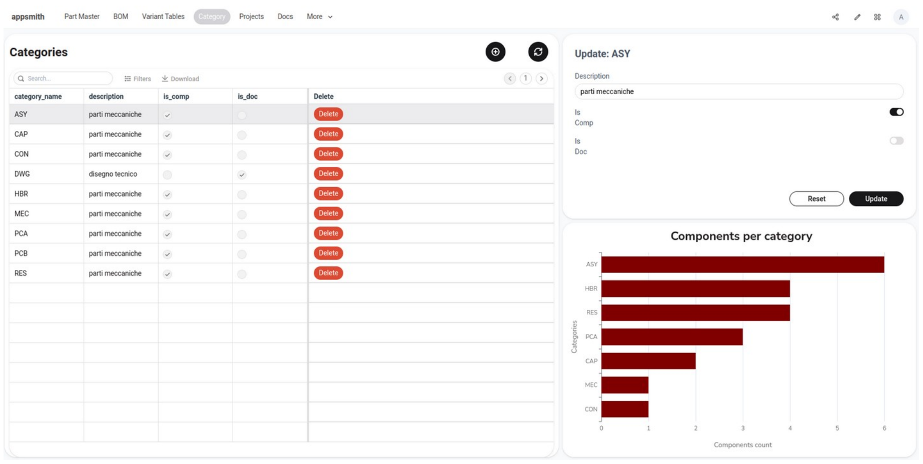
To take full advantage of the flexibility offered by Object-Relational Mapping (ORM), I plan to use the Django framework for implementing my database model. This will enable me to create a reusable and scalable codebase. Additionally, I'll develop a prototype web application with an API to facilitate seamless integration with other applications or models within an ecosystem, enhancing interoperability.

The code is available in a public Github repository `https://github.com/freakontrol/ODM-C`

... code review

Still needs to be implemented the authorization for accessing the data through the API, I should be able to login and generate a personal token to include in every action.
(`https://www.django-rest-framework.org/api-guide/authentication/`)

Furthermore in order to facilitate the data and metadata access by humans, a set of web pages with forms and data visualizations should be implemented in the web app, accessing the same data model of the API.
For example:
`http://esame.cloudcomputing.frk/applications/appsmith/`



Categories

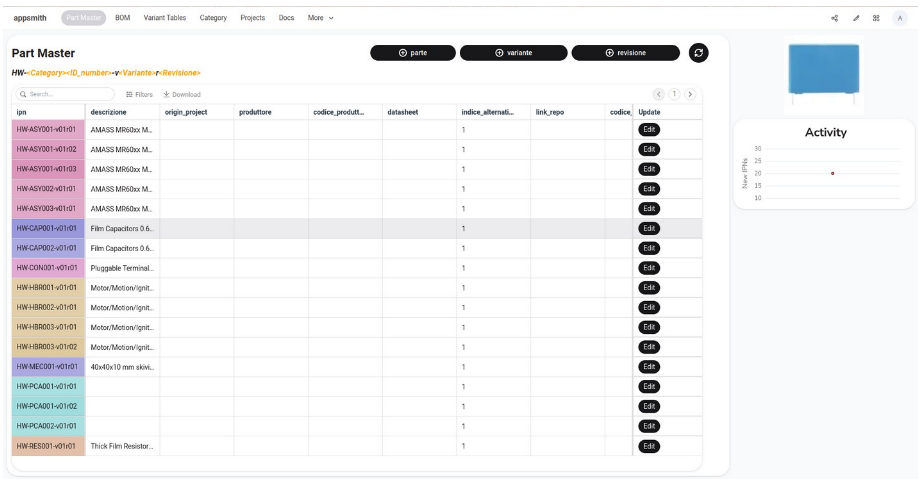
category_name	description	is_comp	is_doc	Delete
ASY	parti meccaniche	✓		Delete
CAP	parti meccaniche	✓		Delete
CON	parti meccaniche	✓		Delete
DWG	disegno tecnico		✓	Delete
HBR	parti meccaniche	✓		Delete
MEC	parti meccaniche	✓		Delete
PCA	parti meccaniche	✓		Delete
PCB	parti meccaniche	✓		Delete
RES	parti meccaniche	✓		Delete

Update: ASY

Description: parti meccaniche

Is Comp ●

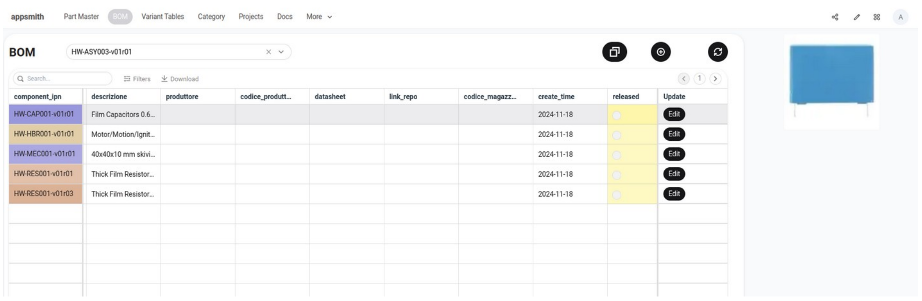
Is Doc ○



Part Master

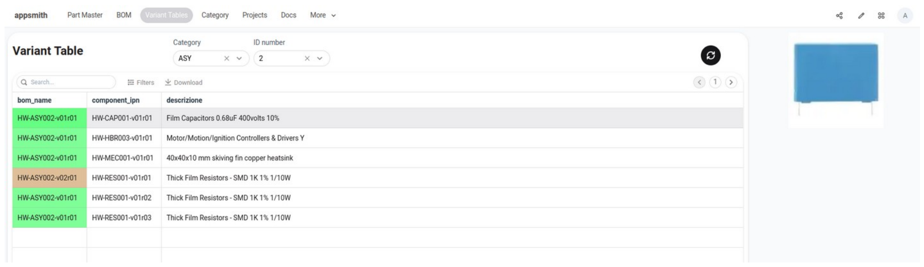
HW-<Category><ID_number>-v<Variante>r<Revisione>

ipn	descrizione	origin_project	produttore	codice_produtt...	datasheet	indice_alternati...	link_repo	codice_	Update
HW-ASY001-v01r01	AMASS MR60xx M...					1			Edit
HW-ASY001-v01r02	AMASS MR60xx M...					1			Edit
HW-ASY001-v01r03	AMASS MR60xx M...					1			Edit
HW-ASY002-v01r01	AMASS MR60xx M...					1			Edit
HW-ASY003-v01r01	AMASS MR60xx M...					1			Edit
HW-CAP001-v01r01	Film Capacitors 0.6...					1			Edit
HW-CAP002-v01r01	Film Capacitors 0.6...					1			Edit
HW-CON001-v01r01	Pluggable Terminal...					1			Edit
HW-HBR001-v01r01	Motor/Motion/Ignit...					1			Edit
HW-HBR002-v01r01	Motor/Motion/Ignit...					1			Edit
HW-HBR003-v01r01	Motor/Motion/Ignit...					1			Edit
HW-HBR003-v01r02	Motor/Motion/Ignit...					1			Edit
HW-MEC001-v01r01	40x40x10 mm skivi...					1			Edit
HW-PCA001-v01r01						1			Edit
HW-PCA001-v01r02						1			Edit
HW-PCA002-v01r01						1			Edit
HW-RES001-v01r01	Thick Film Resistor...					1			Edit



BOM HW-ASY003-v01r01

component_ipn	descrizione	produttore	codice_produtt...	datasheet	link_repo	codice_magazz...	create_time	released	Update
HW-CAP001-v01r01	Film Capacitors 0.6...						2024-11-18		Edit
HW-HBR001-v01r01	Motor/Motion/Ignit...						2024-11-18		Edit
HW-MEC001-v01r01	40x40x10 mm skivi...						2024-11-18		Edit
HW-RES001-v01r01	Thick Film Resistor...						2024-11-18		Edit
HW-RES001-v01r03	Thick Film Resistor...						2024-11-18		Edit



Variant Table

Category: ASY
ID number: 2

bom_name	component_ipn	descrizione
HW-ASY002-v01r01	HW-CAP001-v01r01	Film Capacitors 0.68uF 400volts 10%
HW-ASY002-v01r01	HW-HBR003-v01r01	Motor/Motion/Ignition Controllers & Drivers Y
HW-ASY002-v01r01	HW-MEC001-v01r01	40x40x10 mm skiving fin copper heatsink
HW-ASY002-v02r01	HW-RES001-v01r01	Thick Film Resistors - SMD 1K 1% 1/10W
HW-ASY002-v01r01	HW-RES001-v01r02	Thick Film Resistors - SMD 1K 1% 1/10W
HW-ASY002-v01r01	HW-RES001-v01r03	Thick Film Resistors - SMD 1K 1% 1/10W

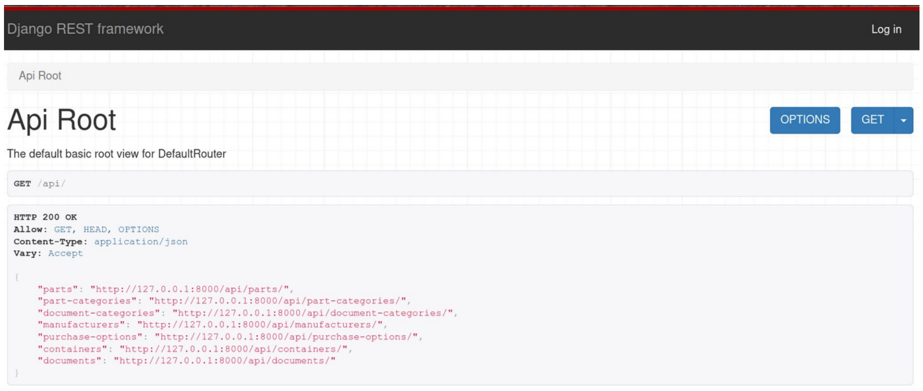
These examples are developed with another tool called Appsmith, it is fast for creating interfaces but is intended for internal applications, not very scalable.
Using Django framework also for the graphical interface should be preferred.

API actions and corrisponding URL schema

Part ViewSet:
List all parts: GET /api/parts/
Create a new part: POST /api/parts/
Retrieve, update or delete a specific part:
GET/PUT/DELETE /api/parts/{internal_part_number}/
Add document to a part:
POST /api/parts/{document_number}/add_document/
Remove document from a part:
POST /api/parts/{document_number}/remove_document/
Create a new item number: POST /api/parts/create_new_item_number/
Create a new variant: POST /api/parts/create_new_variant/
Create a new revision: POST /api/parts/create_new_revision/

PartCategory ViewSet:
List all part categories: GET /api/part-categories/
Create a new part category: POST /api/part-categories/
Retrieve, update or delete a specific part category:
GET/PUT/DELETE /api/part-categories/{name}/
DocumentCategory ViewSet:
List all document categories: GET /api/document-categories/
Create a new document category: POST /api/document-categories/
Retrieve, update or delete a specific document category:
GET/PUT/DELETE /api/document-categories/{name}/
Manufacturer ViewSet:
List all manufacturers: GET /api/manufacturers/
Create a new manufacturer: POST /api/manufacturers/
Retrieve, update or delete a specific manufacturer:
GET/PUT/DELETE /api/manufacturers/{name}/
PurchaseOption ViewSet:
List all purchase options: GET /api/purchase-options/
Create a new purchase option: POST /api/purchase-options/
Retrieve, update or delete a specific purchase option:
GET/PUT/DELETE /api/purchase-options/{pk}/
Container ViewSet:
List all containers: GET /api/containers/
Create a new container: POST /api/containers/
Retrieve, update or delete a specific container:
GET/PUT/DELETE /api/containers/{pk}/
Document ViewSet:
List all documents: GET /api/documents/
Create a new document: POST /api/documents/
Retrieve, update or delete a specific document:

GET/PUT/DELETE /api/documents/{document_number}/



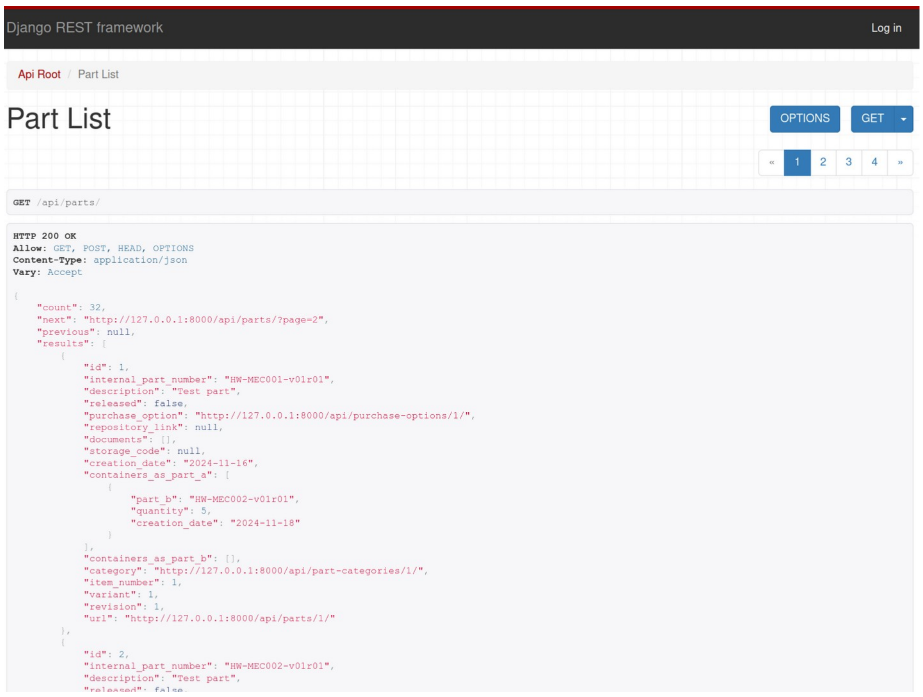
Api Root

The default basic root view for DefaultRouter

```
GET /api/
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "parts": "http://127.0.0.1:8000/api/parts/",
    "part-categories": "http://127.0.0.1:8000/api/part-categories/",
    "document-categories": "http://127.0.0.1:8000/api/document-categories/",
    "manufacturers": "http://127.0.0.1:8000/api/manufacturers/",
    "purchase-options": "http://127.0.0.1:8000/api/purchase-options/",
    "containers": "http://127.0.0.1:8000/api/containers/",
    "documents": "http://127.0.0.1:8000/api/documents/"
}
```

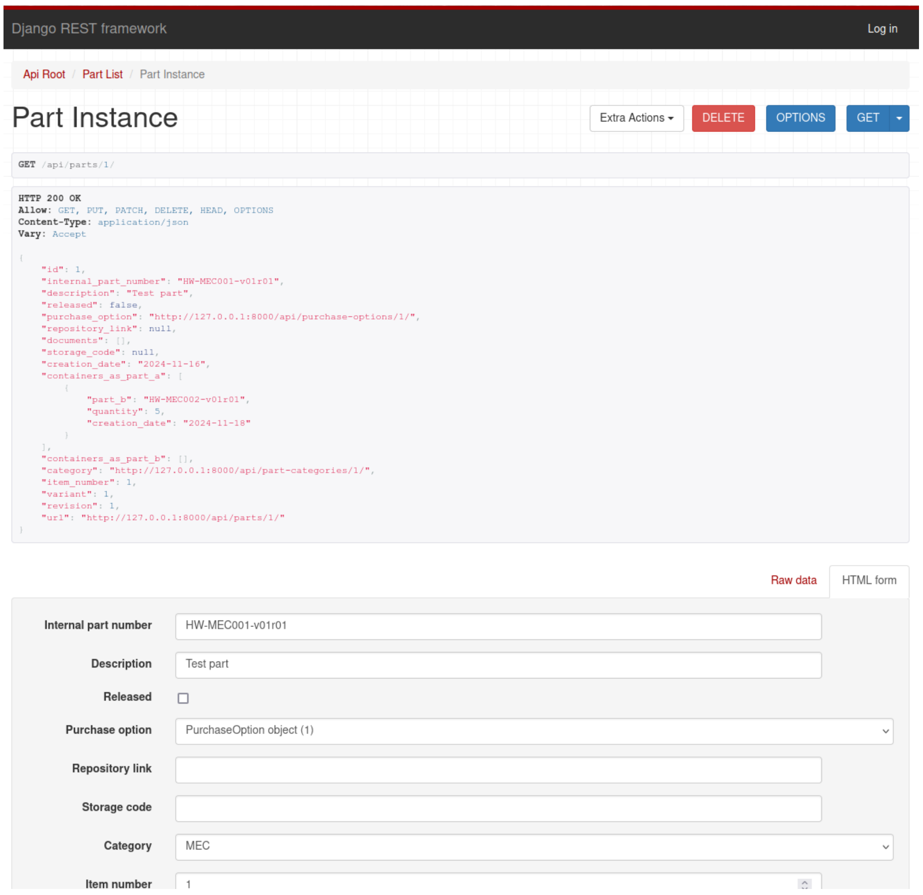


Part List

```
GET /api/parts/
```

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "count": 32,
    "next": "http://127.0.0.1:8000/api/parts/?page=2",
    "previous": null,
    "results": [
        {
            "id": 1,
            "internal_part_number": "HW-MEC001-v01r01",
            "description": "Test part",
            "released": false,
            "purchase_option": "http://127.0.0.1:8000/api/purchase-options/1/",
            "repository_link": null,
            "documents": [],
            "storage_code": null,
            "creation_date": "2024-11-16",
            "containers_as_part_a": [
                {
                    "part_b": "HW-MEC002-v01r01",
                    "quantity": 5,
                    "creation_date": "2024-11-18"
                }
            ],
            "containers_as_part_b": [],
            "category": "http://127.0.0.1:8000/api/part-categories/1/",
            "item_number": 1,
            "variant": 1,
            "revision": 1,
            "url": "http://127.0.0.1:8000/api/parts/1/"
        },
        {
            "id": 2,
            "internal_part_number": "HW-MEC002-v01r01",
            "description": "Test part",
            "released": false,
```



Part Instance

```
GET /api/parts/1/
```

```
HTTP 200 OK
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 1,
    "internal_part_number": "HW-MEC001-v01r01",
    "description": "Test part",
    "released": false,
    "purchase_option": "http://127.0.0.1:8000/api/purchase-options/1/",
    "repository_link": null,
    "documents": [],
    "storage_code": null,
    "creation_date": "2024-11-16",
    "containers_as_part_a": [
        {
            "part_b": "HW-MEC002-v01r01",
            "quantity": 5,
            "creation_date": "2024-11-18"
        }
    ],
    "containers_as_part_b": [],
    "category": "http://127.0.0.1:8000/api/part-categories/1/",
    "item_number": 1,
    "variant": 1,
    "revision": 1,
    "url": "http://127.0.0.1:8000/api/parts/1/"
}
```

Internal part number	HW-MEC001-v01r01
Description	Test part
Released	☐
Purchase option	PurchaseOption object (1)
Repository link	
Storage code	
Category	MEC
Item number	1

6 Cloud Solution Evaluation

Azure App Service is a suitable platform for hosting your Django web app API due to its scalability, reliability, security measures, cost-effective pricing model, ease of use, integration capabilities, and comprehensive documentation.

A tutorial on how to host a Django webapp can be found `here`.