
UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

# EXTENSION OF FUNCTIONALITIES TO THE INVENTORY MANAGEMENT SERVICE PARTKEEPR ON DOCKER

Autor

Luis González Dote

Tutor

Andrés Roldán Aranda


Escuela Técnica Superior de Ingeniería Informática y de Telecomunicación

—

Granada, Septiembre 2025

# AMPLIACIÓN DE FUNCIONALIDADES AL SERVICIO DE GESTIÓN DE INVENTARIO PARTKEEPR SOBRE DOCKER

Luis González Dote

**Palabras clave**: *Partkeepr, software, docker, inventario, componentes electrónicos, documentación técnica, ampliación de funcionalidades, guía instalación, despliegue.*

**Resumen**

PartKeepr es un software libre para gestionar componentes electrónicos. Se utiliza en laboratorios, makerspaces y pequeñas empresas de hardware por su capacidad para catalogar piezas, controlar existencias y generar listas de materiales. El problema está en que la versión actual presenta carencias de usabilidad, control de acceso y, sobre todo, no tiene documentación técnica.

El trabajo se centra en dos frentes complementarios. Por un lado, se estandariza la puesta en marcha con Docker Compose, separando servicios (aplicación, base de datos y servidor web), definiendo volúmenes persistentes y añadiendo comprobaciones. Se documentan procedimientos de instalación limpia, restauración y copia de seguridad para que cualquier equipo pueda replicar el entorno sin depender de máquinas concretas ni configuraciones a mano. Por otro lado, se refuerza la aplicación desde dentro con cambios que mejoran seguridad, usabilidad y mantenimiento.

Entre las mejoras funcionales, destaca un modelo de permisos granular que separa claramente quién puede crear o borrar componentes del rol de administración, reduciendo riesgos operativos y haciendo el sistema más predecible. Los perfiles de usuario incorporan teléfono y fotografía (con miniatura) para facilitar la identificación y la comunicación interna. La navegación diaria es más ágil gracias a las miniaturas en adjuntos, el resaltado no destructivo en los resultados de búsqueda y un buscador en el árbol de categorías que evita recorrer nodos de forma manual. Para tareas repetitivas, se habilita la edición directa de stock y precio desde la rejilla, respetando siempre los permisos definidos. Además, se integra Nexar como fuente externa para consultar datos de componentes y completar fichas sin salir del sistema.

Todo lo anterior se acompaña de una radiografía de la arquitectura: bundles, modelo de datos, flujos y puntos de extensión, de modo que incorporar cambios resulte más seguro y previsible.

Con todo esto se pretende extender la vida útil de PartKeepr, reducir su deuda técnica y mejorar la experiencia de usuario, aportando a la comunidad una herramienta de inventario robusta, segura y plenamente documentada.

# EXTENSION OF FUNCTIONALITIES TO THE PARTKEEPR INVENTORY MANAGEMENT SERVICE ON DOCKER

Luis González Dote

**Keywords**: *PartKeepr, software, Docker, inventory, electronic components, technical documentation, feature expansion, installation guide, deployment.*

**Abstract**

PartKeepr is free software for managing electronic components. It is used in laboratories, makerspaces, and small hardware companies for its ability to catalog parts, control inventory, and generate bills of materials. The problem is that the current version has shortcomings in usability, access control, and, above all, lacks technical documentation.

The work focuses on two complementary fronts. On the one hand, startup is standardized with Docker Compose, separating services (application, database, and web server), defining persistent volumes, and adding checks. Clean installation, restoration, and backup procedures are documented so that any team can replicate the environment without relying on specific machines or manual configurations. On the other hand, the application is reinforced from within with changes that improve security, usability, and maintenance.

Among the functional improvements, a granular permissions model stands out, which clearly separates who can create or delete components from the administration role, reducing operational risks and making the system more predictable. User profiles incorporate a phone number and photo (with thumbnail) to facilitate identification and internal communication. Daily navigation is more agile thanks to thumbnails in attachments, non-destructive highlighting in search results, and a search engine in the category tree that avoids manually navigating nodes. For repetitive tasks, direct editing of stock and price from the grid is enabled, always respecting the defined permissions. In addition, Nexar is integrated as an external source for consulting component data and completing files without leaving the system.

All of the above is accompanied by an in-depth analysis of the architecture: bundles, data model, flows, and extension points, so that incorporating changes is safer and more predictable.

The aim of all this is to extend the useful life of PartKeepr, reduce its technical debt, and improve the user experience, providing the community with a robust, secure, and fully documented inventory tool.

Yo, **Luis González Dote**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingeniería Informática y de Telecomunicación de la Universidad de Granada**, con DNI 75934191J, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Luis González Dote

GONZALEZ DOTE LUIS - 75934191J

Firmado digitalmente por GONZALEZ DOTE LUIS - 75934191J
Fecha: 2025.09.05 08:29:33 +02'00'

Granada a 05 de Septiembre de 2025

D. **Andrés Roldán Aranda**, Profesor del Departamento Electrónica y Tecnología de Computadores de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado **Ampliación de funcionalidades al servicio de gestión de inventario PartKeepr sobre docker**, ha sido realizado bajo su supervisión por Luis González Dote, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 05 de Septiembre de 2025

Los directores:

ROLDAN ARANDA ANDRES MARIA - 26487749Y

Firmado digitalmente por ROLDAN ARANDA ANDRES MARIA - 26487749Y
Fecha: 2025.09.05 08:32:02 +02'00'

# Agradecimientos

Quiero agradecer a mi familia, por brindarme esa confianza y apoyo constante que ha sido fundamental a lo largo de mis estudios. Gracias por recordarme cada día que no habrá éxito sin esfuerzo ni dedicación, vuestra paciencia, ejemplo y consejos han sido el motor que me ha ayudado a seguir adelante.

# 1 Index

# Index of figures

# 1. Introduction

## 1.1 Context and motivation

### 1.1.1 Context

In any electronics workshop, from a university workbench to a small company's storeroom, keeping track of every resistor, microcontroller or connector can quickly become a nightmare. PartKeepr was created precisely to bring order to that chaos. It is a free application that allows components to be registered, stock to be monitored, and bills of materials to be prepared without depending on proprietary software.

Over the years, it has earned a place in laboratories, makerspaces and assembly companies, but the tool has gradually fallen short in several areas:

- **Permissions that are too basic.** Any user can delete or create components, something unthinkable when several people manage the same inventory.
- **Usability in need of improvement.** Profiles display little more than a name, with no photos or phone numbers to help identify the person in charge. Moreover, navigating large catalogues is slow, and updating common data, such as stock levels, requires too many clicks.
- **Lack of technical documentation.** New contributors face code without manuals, discouraging collaboration and increasing maintenance costs.

Meanwhile, more recent projects such as InvenTree, or paid services like PartsBox, already offer fast search engines, online editing, and detailed guides. If PartKeepr wishes to remain the leading free option, it needs to be brought up to date.

### 1.1.2 Motivation

This work is conceived as a practical and low-risk intervention to extend the useful life of PartKeepr and make its maintenance easier.

Operational security is reinforced with a granular permission scheme that distinguishes between administration, creation, and deletion of components, while the interface is aligned to show only what each profile is authorised to use.

Everyday use is improved through small but impactful changes such as thumbnails in attachments, non-destructive highlighting in the grid to enable searching without losing context, direct and permission-controlled editing of stock and price, and a category-tree search tool that saves time in workshops with numerous components and folders.

External integration is updated by replacing the old connection with Octopart with Nexar, preserving the part-number search experience without altering the data model.

Additionally, the uniqueness of sub-locations is corrected to allow repeated names in different branches of the tree, as required by real-world use. All of this is accompanied by a reproducible deployment with Docker/Compose and an architectural guide explaining how the project is organised, how data flows, and where it can be safely extended.

## 1.2 Objectives

### 1.2.1 General objective

Develop and document an improved version of PartKeepr that increases the security, usability, and maintainability of the platform, so that it can continue to be an open-source benchmark for electronic inventory management in academic and industrial environments.

### 1.2.2 Specific objectives

| Name | Granular permissions by role |
|---|---|
| Description | Replace the 'all-or-nothing' scheme with differentiated permissions for administration, creation, and deletion of components. Reflect these permissions in the UI and backend. |
| Succes indicator | Addition, removal, and duplication of components 100% protected by permissions. User editing (permission checkboxes) visible only to administrators. 403 responses for unauthorised attempts and disabled controls in the interface. |

| Name | Enriched user profiles |
|---|---|
| Description | Add telephone number and photo to user profile, with upload from UI, thumbnail and default image. |
| Succes indicator | Data visible in the user profile and served by the API. Thumbnail displayed in the interface. |

| Name | Thumbnails in attachments |
|---|---|
| Description | Display thumbnails in attachments associated with a component to speed up the identification of technical images. |
| Succes indicator | Rendering images in the attachments table with no noticeable impact on performance. |

| Name | Dynamic search engine in the category tree |
|---|---|
| Description | Add a dynamic search engine to the category tree that filters large catalogue folders in real time. |
| Succes indicator | Average time to locate a category reduced by ≥ 40%. |

| Name | Online editing of stock and average price |
|---|---|
| Description | Enable online editing of stock and average price fields directly in the components table, only if the user has permissions. |
| Succes indicator | Update values with a single click and validate permissions accordingly. |

| | |
|---|---|
| Name | Non-destructive highlighting in the search grid |
| Description | Highlight matches in visible columns without filtering the store or altering the order or counters. |
| Succes indicator | Terms highlighted in yellow, with no changes to pagination or groupings. Highlighting disappears when the search field is cleared. |

| | |
|---|---|
| Name | Integration with Nexar for component consultation |
| Description | Integration with Nexar (replacing Octopart). Consult Nexar GraphQL for MPN search and complete data sheet retrieval (manufacturer, distributors, prices, documents, images). |
| Succes indicator | Paginated list of results. |

| | |
|---|---|
| Name | Composite uniqueness in storage sub-locations |
| Description | Allow duplicate StorageLocation names in different branches of the tree, ensuring uniqueness only within the same category. |
| Succes indicator | Creation of ‘Bar 1’ in two different categories permitted; duplication within the same category rejected |

| | |
|---|---|
| Name | Technical architecture guide |
| Description | Draft the technical architecture guide: bundles, data flows, and extension points. |
| Succes indicator | Section included in the report |

| | |
|---|---|
| Name | Deployment and migration manual with Docker |
| Description | Develop a deployment and migration manual with Docker-Compose. |
| Succes indicator | Reproducible installation in a clean environment by following the step-by-step guide. |

## 1.3 Structure of the document

This section explains how the thesis has been organised so that it can be read from the outside in: first the problem and context, then the technical details, and finally the tests, deployment, and conclusions.

The journey begins with the introduction, where the context and motivation are outlined and the objectives that guide the work are formulated. This provides the reader with a clear map of what to expect next.

Subsequently, the project management section presents the time planning, the resources used, and the estimated budget. It shows how the work was divided into blocks, which deliverables were established, and with what means they were executed, ensuring traceability is clear and making it easy to understand the progress.

The state of the art offers an overview of current solutions for electronic inventory management, ranging from installable open-source platforms to SaaS services, and helps identify the specific gap that this project addresses. This comparative framework serves both to justify the technical and product decisions taken later on and to position PartKeepr against more recent alternatives.

On this basis, the document moves into a detailed analysis of the application in the study of PartKeepr's architecture. The bundle and dependency structure, the data model, and the information flows are described, together with the extension points and recommended practices for safely evolving the system. The chapter closes with architectural conclusions that summarise strengths and areas for improvement, thus preparing the ground for practical interventions.

From there, the functional improvements are developed in blocks: the granular permission model, why it was needed, how it was designed, and what changes in operation it introduces, the enrichment of user profiles with phone number and photo, thumbnails in attachments, integration with Nexar as a replacement for Octopart, resolution of uniqueness conflicts in sub-locations, inline editing control in the grid for stock and price, and the new search functions (non-destructive highlighting in the list and search in the category tree). Each section follows the same structure, context, design/implementation, and results, so that reading is consistent and verification straightforward.

Finally, the deployment and dockerisation of PartKeepr is documented, with differentiated procedures for clean installation and restoration. An additional chapter is included on updating and another on backup and rapid recovery, aimed at everyday operation. The thesis concludes with the Conclusions and the Bibliography and References, which close the cycle with a synthesis of results and the consulted sources; where relevant, guides, scripts, and evidence are included in the appendices.

# 2 Project management

## 2.1 Time Planning

The project schedule is summarised in a Gantt chart covering the period from November 2024 to August 2025. [1]

It began with a short initiation phase and the state of the art, followed by an architectural study that extended until mid-January. This stage was expanded beyond the initial plan as it provided key evidence for the rest of the work.

From that point onwards, the development and improvement block continued until the end of May, with short and overlapping iterations to reduce waiting times. Testing was not postponed until the end; it started as soon as the first version of granular permissions was ready and accompanied each iteration through to the final change.

Dockerisation and deployment were carried out between late May and early June. Writing and preparation for the defence took place from June to mid-August, interspersed with revisions and refinement of the guides. Each milestone generated a verifiable deliverable (functionality, tests, or guide), making it possible to track the actual progress of the project at a glance in the Gantt chart.

| Nombre de tarea | Fecha de inicio | Fecha final |
|---|---|---|
| | **2024/11/11** | **2025/09/01** |
| **Planning** | **2024/11/11** | **2024/11/14** |
| Scope & objectives | 2024/11/11 | 2024/11/13 |
| Work plan & schedule | 2024/11/11 | 2024/11/14 |
| **State of the Art** | **2024/11/18** | **2024/11/27** |
| Literature search & review: | 2024/11/18 | 2024/11/25 |
| Synthesis & writing | 2024/11/25 | 2024/11/27 |
| **Architecture Study** | **2024/12/09** | **2025/01/17** |
| Data model & flows | 2024/12/09 | 2024/12/20 |
| Bundles & dependencies map | 2024/12/13 | 2025/01/10 |
| API surface & security | 2024/12/18 | 2025/01/17 |
| **Development & Enhancements** | **2024/12/05** | **2025/05/30** |
| Granular permissions | 2024/12/05 | 2024/12/20 |
| User profiles (phone/photo + thu... | 2024/12/12 | 2025/01/20 |
| Thumbnails for attachments | 2024/12/19 | 2025/01/09 |
| Nexar integration | 2024/12/18 | 2025/03/31 |
| Storage sub-location uniqueness | 2025/01/10 | 2025/01/31 |
| Inline edit for stock & price | 2025/01/15 | 2025/02/17 |
| In-grid highlighting | 2025/02/03 | 2025/02/28 |
| Category tree search | 2025/02/17 | 2025/03/17 |
| Polish & small fixes | 2025/04/01 | 2025/05/30 |
| **Testing & QA** | **2024/12/20** | **2025/05/30** |
| Feature testing & regression suite | 2024/12/20 | 2025/05/30 |
| **Dockerization & Deployment** | **2025/05/22** | **2025/09/01** |
| Docker/Compose stack (app, db, ... | 2025/05/22 | 2025/05/29 |
| Deployment / restore guide | 2025/09/01 | 2025/09/01 |
| Install & backup scripts | 2025/05/30 | 2025/06/03 |
| Dockerization & Deployment | 2025/06/03 | 2025/06/06 |
| **Writing & Defense Prep** | **2025/06/02** | **2025/08/22** |
| Writing (full draft) | 2025/06/02 | 2025/06/30 |
| Review & final edits | 2025/07/01 | 2025/07/31 |
| Writing & Defense Prep | 2025/08/11 | 2025/08/22 |

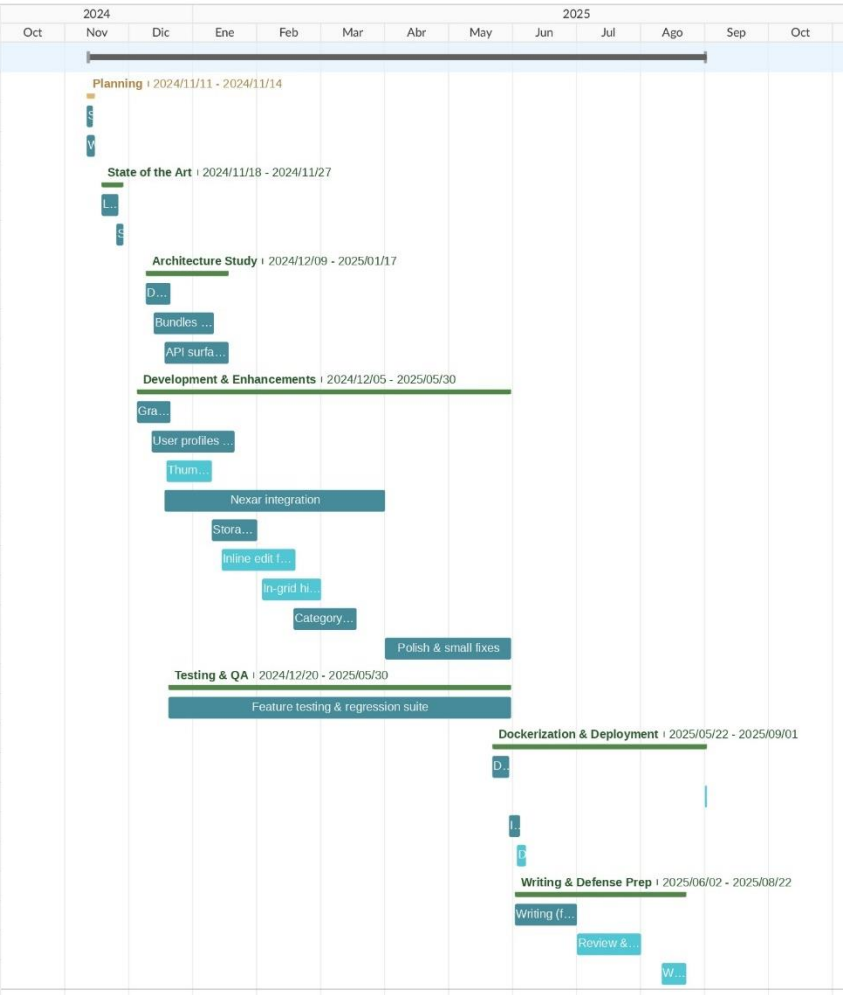



*Figure 1. Gantt chart*

## 2.2 Resources and estimated budget

This project was carried out with an average dedication of six hours per day. To estimate the budget, a conservative and reproducible approach was adopted:

1. The personnel cost was first calculated based on a reference salary for a junior profile in Spain.
2. Reasonable costs of hardware actually used were included (with partial depreciation where applicable).
3. Costs of software and services were quantified (when not free or involving significant usage).
4. Other operating expenses were added (electricity, connectivity, backups).

5. A contingency of 10% was applied to cover minor unforeseen events.

This method makes the assumptions explicit and facilitates recalculating if rates or scope change.

### 2.2.1 Personal cost

As salary reference, the usual range for a junior programmer in Spain of €18,000–€25,000 gross per year was taken, based on recent compilations from InfoJobs, Glassdoor, and Tecnoempleo, and summarised, among others, by KeepCoding (average net monthly salary €1,500–€1,800 depending on city and company) [2]

For the calculation, an intermediate point of €22,000 gross per year (12 payments) was fixed, equivalent to €1,833 gross per month. [3] The actual company cost includes contributions and social security charges.

In projects of this size, an overhead of 30–35% over the gross monthly salary is usually budgeted. A rate of +32% was adopted, which sets the company's monthly cost at €2,417. With a standard working day of 160 h/month, the hourly cost amounts to €15.1.

The effective planning of this project resulted in 765 hours of work (an average of 6 h/day on effective working days within the period). The estimated personnel cost therefore amounts to:

765 h × €15.1/h = **€11,565**

This figure reflects a conservative scenario for a junior profile and serves as a basis for comparison should more staff become involved in the future or salaries change.

### 2.2.2 Hardware cost

Only equipment and peripherals strictly required for development and testing are charged, applying proportional depreciation according to usage during the project.

- Development laptop (mid-range, €1,000): 20% allocation for intensive use during the thesis → €200.
- Auxiliary monitor (€150): 50% allocation → €75.

- External 1 TB SSD for local backups (€100): 100% allocation → €100.
- Smartphone for basic interface/web testing (€230): 15% allocation → €34.5.
- Minor systems (stand, cables, USB hub, etc.): €35 (estimate).

**Total hardware allocated: €445**

### 2.2.3 Software cost

The development relied on free or open-source tools, or those with free licences for academic/personal use:

- Docker, Nginx, MariaDB, PHP and extensions, Composer, VS Code.

The only recurring expense arises from the external component search through Nexar (the replacement for the former Octopart). To operate its API in production, a subscription plan is required.

The Standard plan is priced at €100/month and covers moderate volumes (approx. up to 2,000 new components matched per month), while the Pro plan rises to €500/month and is designed for high loads (around 15,000 components/month).

The choice depends on the actual rate of new entries in PartKeepr. If the laboratory incorporates only a few references each month, the Standard plan is sufficient. If entries are automated or entire catalogues are imported, the Pro plan prevents bottlenecks.

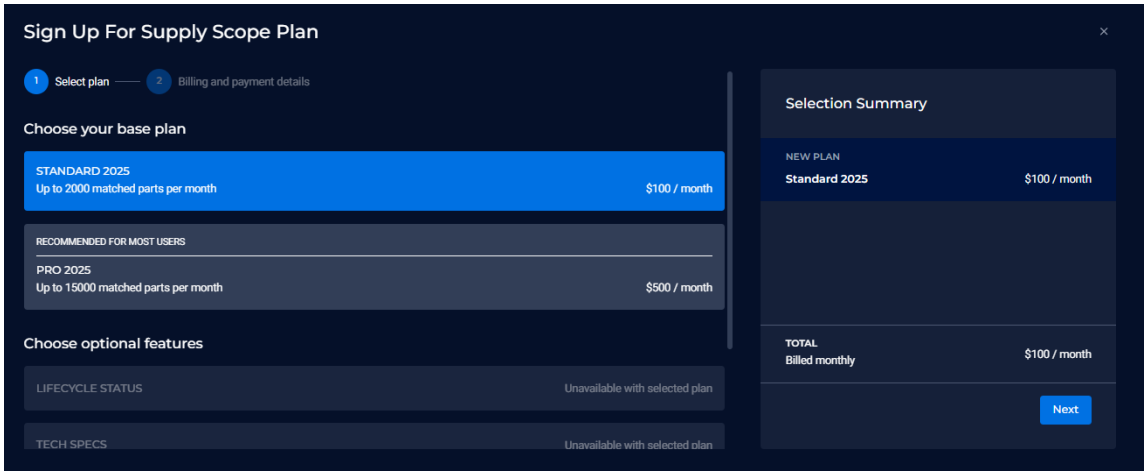



*Figure 2. API Plans Nexar*

Therefore, the summary of software expenses is as follows:

- Base software (Docker, Nginx, PHP, MariaDB, Composer, Ext JS GPL): €0
- Nexar (Standard plan, 1 month for commissioning): €**100**

### 2.2.4 Other cost and contingency

**Electricity**: equipment averaging 120 W × 6 h/day × 128 effective days = 92 kWh. At €0.22/kWh → €20

- **Connectivity**: partial allocation of home tariff (€30/month × 9.5 months × 20% usage) → €57
- **Backups** (cloud space/basic service): €18
- Office supplies and occasional printing: €10

**Subtotal other costs: €105**

Given the technical nature of the project, a 10% contingency is added to the subtotal (personnel + hardware + software + other) to absorb minor overruns (replacement of peripherals, shipping costs, etc.)

### 2.2.5 Budget summary

| Item | Amount |
|---|---|
| Personnel costs (765 hours) | 11.565€ |
| Allocated hardware | 445€ |
| Software | 0€ |
| Other operating costs | 105€ |
| Subtotal | 12.115€ |
| Contingency (10%) | 1.212€ |
| **Total estimated budget** | **13.327€** |

The budget can be easily recalculated if assumptions change (for example, using €25,000 gross annual salary or applying 35% social costs). In that case, the hourly cost would increase and the total would move to the €14–15k range for the same number of hours. This calculation, documented step by step, allows each parameter to be adjusted transparently in future revisions.

# 3 State of the art

Electronic component inventory today lies at a crossroads between self-hosted free software, fully cloud-based solutions, and hybrid options with connectors to CAD/PLM tools. Broadly speaking, three families coexist:

- Open-source platforms installable on one's own server
- Cloud services with lightweight clients or plug-ins
- SaaS solutions specialised in BOM and traceability

The differences are not only a matter of licence or cost. They also affect deployment speed, the level of automation possible, and the degree of integration into the daily work of a workshop, laboratory, or small production line.

**Open-source platforms**


- **PartKeepr** established itself for years as the specific free alternative for electronics: detailed part records, tree-based categories, attachments, and stock history. Its public repository clearly reflects this scope, although its development pace has slowed, which explains part of the technical debt addressed in this work. Even so, the functional base (Symfony 2.8, Doctrine, ExtJS client) and its focus on electronics remain a differentiating value for many users who prioritise data sovereignty and local control of the system. [4]


- **InvenTree** represents the more recent generation: a Python/Django backend, native REST API, and deployment designed from the outset for containers. The official documentation covers installation with Docker/Compose, auxiliary processes, and a permission model for users and groups, making automation and CI/CD easier in modern environments. The manual itself highlights these aspects (API, permissions, deployment options), aligned with current expectations in collaborative environments. [5]

ERPNext

- **ERPNext**, in turn, offers far more than inventory: it is a complete ERP with stock, purchasing, sales, and manufacturing modules. The documentation of the stock module illustrates well this generalist approach, replenishment rules, units of measure, multiple warehouses. This breadth is powerful when inventory cannot stand alone and must coexist with accounting or procurement, though it introduces complexity that may be excessive if the sole need is to manage electronic components. [6]

**Hybrid services (cloud + plug-ins)**

openbom

- **OpenBOM** is a cloud-based PLM solution developed by Newman Cloud (California), which centralises product data and the processes associated with its lifecycle, from idea to manufacturing. It is offered as a 100% cloud service (no in-house infrastructure required) and includes onboarding materials, tutorials, and videos to reduce the learning curve.

  Its proposal revolves around a common repository of parts, bills of materials (BOM), prices, suppliers, and revisions, with real-time collaboration between engineering, procurement, and production. In daily practice, this translates into fewer errors due to duplicate information, simpler workflows, and more efficient operation that helps cut costs and accelerate deliveries. [7]

  The platform's differential value is based on three pillars:

  - A carefully designed, easy-to-use interface.
  - Direct integration with the most common CAD programs (e.g. SolidWorks or Autodesk Inventor, among others), which avoids manual exports and keeps BOMs synchronised.
  - Change and revision management that leaves a clear trail of who modified what and when.

  As a natural trade-off for a SaaS service, the data resides on the provider's infrastructure and certain advanced features are part of subscription plans, so it is advisable to evaluate internal requirements (data sovereignty, integration with existing systems) before adoption. Overall, OpenBOM is a solid option for those looking for lightweight PLM, ready-to-use integrations, and a quick start without setting up their own servers.

**Saas services**


- **PartsBox** is a web service for managing electronic inventory and BOMs without struggling with spreadsheets. It centralises which components are available, where they are, and how much remains, and it also indicates what can be manufactured with the available stock. From the same interface it manages versioned bills of materials, prepares kits for light production, and assists in purchasing (quantities, prices, orders). In everyday use, fast search, traceability of movements, and practical details such as labels and barcodes are highly valued.

  Being SaaS, it requires no server setup or database maintenance, and it offers an API for task automation. The trade-off is that data is hosted on the provider's cloud and free plans have limits, so it is important to assess whether it fits the team's needs. [8]

**Cross-cutting trends**

Several patterns recur when reviewing these solutions:

1 **Containerised deployment**: active projects publish production-ready guides and Docker stacks, as in the case of Iventree, or facilitate packaging to ensure reproducible installations.

2 **API automation**: having well-documented REST or official connectors allows the inventory to be integrated with purchasing scripts, labelling or assembly stations.

3 **Fine-grained permission control and traceability**: users, groups, and auditing are now the norm in shared environments.

4 **User experience**: fast search that does not break context, thumbnails of attachments and direct editing in tables for frequent tasks, and living documentation that shortens the learning curve (again, the InvenTree guide is illustrative in this regard).

**The gap covered by this project**

In the current landscape, highly polished cloud solutions coexist with modern open-source projects enjoying an active development pace. Between these two extremes there remained a gap: a free, self-hosted tool, designed specifically for electronics, that would offer today's conveniences (granular permissions, fast searches, thumbnails, inline editing) without forcing a complete rewrite or moving data off-site.

This is where PartKeepr had fallen short. It remained useful for its focus and for local inventory control, but suffered from outdated usability, overly basic permissions, and limited technical documentation for onboarding new contributors.

This work addresses precisely that gap. It preserves the value proposition of PartKeepr (free software, data control, and self-deployment) while adding the features now taken for granted, such as a granular and understandable permission model, richer user profiles, thumbnails in attachments, and more.

# 4 Study of the Architecture of PartKeepr

### 4.1.1 General overview of the core (Symfony 2.8)

This project is built on Symfony 2.8 [9] and a classical stack for PHP applications with a REST API. The picture can be obtained directly from the configuration files themselves (AppKernel.php, config.yml, routing.yml, security.yml, config_doctrine.yml, config_dunglas.yml, config_fos_user.yml, config_partkeepr.yml and partkeepr.yml).

The boot cycle is the usual one: AppKernel first registers the base and third-party bundles. In dev and test environments, it adds development tools, and finally loads the functional PartKeepr bundles. It also leaves the door open for customisation with getCustomBundles(). A useful detail for deployment is that the cache directory depends on the PARTKEEPR_ENVIRONMENT variable, which makes it easy to separate environments in Docker.

### 4.1.2 Third-party bundles: What they provide and why they are there

The third-party core covers the typical needs of a mature web application:

- **Infrastructure**: FrameworkBundle, SecurityBundle, TwigBundle, MonologBundle and AsseticBundle provide MVC, security, templates, logging, and asset management, the latter now being legacy and advisable to replace.
- **Persistence:** DoctrineBundle for ORM and StofDoctrineExtensionsBundle for Gedmo extensions (e.g., trees), configured in config_doctrine.yml.
- **Communication**: FOSRestBundle [10] manages the REST layer (request body handling, format negotiation, and its own exception handler partkeepr.exceptionwrapper), while Dunglas\ApiBundle [11] (the classic version of API Platform) publishes resources and operations, including JSON-LD, CSV, and XLSX.

- **Lifecycle support:** DoctrineMigrationsBundle for migrations, DoctrineFixturesBundle for fixtures, and LiipFunctionalTestBundle in dev/test for functional testing.
- **Advanced authentication:** EscapeWSSEAuthenticationBundle (WSSE), FR3DLdapBundle for LDAP, and KnpGaufretteBundle to abstract the file system in handling attachments and images.
- **Front and static resources:** SpriteGeneratorBundle and BrainbitsFugueIconsBundle handle iconography and sprites.

Risks debt observed:

- Assetic and Dunglas are technologies from another era. In the medium term, it is reasonable to plan a migration to Symfony ≥5/6, modern API Platform, and an asset pipeline with Webpack/Vite.

### 4.1.3 PartKeepr`s own bundles: how the domain is divided

The application divides responsibilities by functional areas:

- **Domain core**: CoreBundle (utilities, migrations, system notices), RESTBundle (REST glue), SetupBundle (installation wizard).
- **Identity and security**: AuthBundle (FOSUser users, providers, login and password-change actions).
- **Catalogue and categories**: PartBundle (Part entity and stock operations), StockBundle (stock entries), CategoryBundle (trees with Gedmo and move/root node operations).
- **Manufacturers and suppliers**: ManufacturerBundle and DistributorBundle, with support for logos/images.
- **Physical storage**: StorageLocationBundle (locations, categories, and images).
- **Footprint and attachments**: FootprintBundle, ImageBundle (temporary uploads, webcam upload, thumbnails) and UploadedFileBundle (file lifecycle).
- **Interface**: FrontendBundle and MobileFrontendBundle package the UI in ExtJS.
- **Projects and reports**: ProjectBundle (projects, reports, and attachments).
- **Preferences, import/export, and utilities**: SystemPreferenceBundle, ImportBundle, ExportBundle, StatisticBundle, CronLoggerBundle.
- **External integration**: OctoPartBundle (historic integration with Octopart).

This organisation clearly separates infrastructure and domain. Each bundle exposes its entities and controllers and, when applicable, also Dunglas operations with routes under /api.

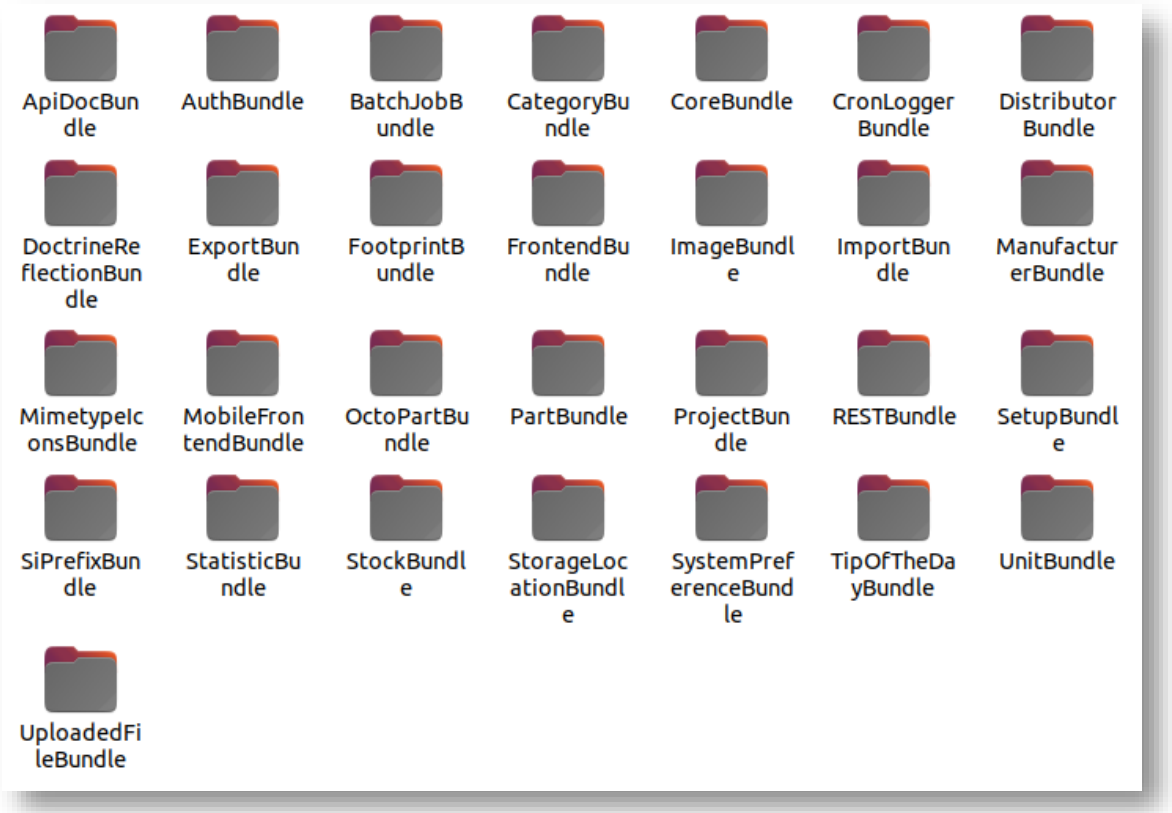



*Figure 3. PartKeepr bundles*

## 4.2 Data flows and entity model

### 4.2.1 Overview of the data model

In PartKeepr, information is organised into domain entities that Doctrine maps to tables. Each entity represents a business concept, such as part, category, or supplier, and is related to others through 1→N, N→1, and M→N cardinalities, allowing for a precise description of how different inventory items interact.

At the heart of the system is the part entity, around which the entire inventory revolves. Every stock movement is recorded in stockentry, so that additions, removals and adjustments are tracked and stock levels can be reliably derived. The technical characteristics of the parts are collected in partparameter and, when composite criteria or ranges are required, they are supported by metapartparametercriteria. Manuals, data sheets and images are associated using partattachment, which links each part to its documentation.

Classification and location are resolved using tree structures to maintain clean and efficient hierarchies. Part categories (partcategory), footprint or package categories (footprintcategory), and physical location categories (storagelocationcategory) are stored as nested sets with lft, rgt, and lvl fields, making it easy to move entire subtrees without losing consistency. The footprint itself can have images and attachments, and each storage location (storagelocation) identifies the physical location where the components are stored, with optional image support for quick recognition.

The external catalogue is modelled through Manufacturer and Distributor, with the option to associate logos with manufacturers (manufacturerICLogo). The partmanufacturer and partdistributor relations link each part with its market references, including manufacturer part number, distributor SKU, price, and currency. This is an M→N relationship that allows multiple sources for a single part and, conversely, multiple parts per supplier.

For projects and reporting, the system incorporates project and projectpart, which describe which parts are used by each project, as well as projectrun and projectrunPart to record specific production runs. The reporting layer is articulated with Report, reportproject, and reportpart, enabling the generation of bills of materials and summaries of purchases or consumption.

## 4.3 Extension points and good development practices

### 4.3.1 Extension map in PartKeepr

With respect to the domain layer, when adding or modifying business rules, entities, or repositories, the appropriate place is within the domain bundles (partbundle, stockbundle, categorybundle, uploadedfilebundle, imagebundle, projectbundle, etc.). The practical guideline is clear: each change must remain within its own domain (for example, stock changes should not be introduced in the frontend).

This organisation can be deduced by mapping the bundles registered in AppKernel.php, locating the entities in src/PartKeepr/**/Entity, and verifying that the annotation-based routing of each bundle appears grouped in routing.yml. The coherence of this structure confirms the classical Symfony 2.x pattern of functionality grouped by bundle. [12]

For the API layer, REST routes are defined declaratively in app/config/config_partkeepr.yml using resource.* services and item_operation/collection_operation operations, for example PUT /parts/{id}/addStock, POST /temp_images/upload. This arrangement is verified by searching for the prefix /api in routing.yml and following the creation of resources and associated controllers in config_partkeepr.yml.

The information visible to the client depends on the @Groups defined in the entities and referenced from the resources. If the frontend does not receive a new field, it is usually because the relevant group does not include it. This can be confirmed by reading, in each resource. of config_partkeepr.yml, the calls to initNormalizationContext/initDenormalizationContext (where groups are listed) and comparing them with the annotations in the entities. This defines the explicit contract between the server and ExtJS.

For attachments and images, the file abstraction (Gaufrette) and the temporary→final workflow orchestrated by uploadedfilebundle/imagebundle must be used instead of manual copying. This conclusion is drawn from the temporary listeners defined in config_partkeepr.yml and the physical paths in partkeepr.yml, which should also be mounted as Docker volumes to guarantee persistence.

### 4.3.2 Typical Extension Points (Established Patterns)

New business operation on Part, such as reserving stock.

The recommended pattern is to declare the operation in config_partkeepr.yml, for example, PUT /parts/{id}/reserveStock, implement the controller action and, if applicable, complete with a listener. This recipe is inferred from the existing operations (addStock, removeStock, setStock) defined in resource.part, whose structure serves as a guide.

New attachment type. The established operational workflow is reused: upload to TempUploadedFile/TempImage (collection POST operations), link to the target entity (Attachment), and automatic move to the final folder via a kernel.view listener. All of this can be seen in resource.tempfile/resource.tempimage and in the entities PartAttachment, FootprintAttachment, and ProjectAttachment.

Extending technical parameters of parts. If greater detail is required, fields can be added to PartParameter (or to an auxiliary entity), @Groups adjusted, the corresponding migration created, and the ExtJS store updated. This strategy is inferred from the current model (parameters, units, and SI prefixes) and the groups in use.

## 4.4 Conclusions of the Architectural Analysis

After reviewing code and configuration, it is evident that PartKeepr follows a classic Symfony 2.8 + Doctrine scheme with ExtJS client and declarative API. The bundle structure allows each responsibility to be located without invading other areas, and listeners ensure consistency (stock, category trees, and file lifecycle) without depending on the client. Responsibilities are distributed into coherent bundles (for example, partbundle for parts, authbundle for authentication, corebundle for cross-cutting utilities), which makes it easier to locate change points without affecting other areas.

When it comes to Partkeepr's strengths, the architecture shows clear modularity by domain, allowing you to quickly locate the exact point where you need to intervene without affecting other areas. The API is consistent and predictable, with well-defined resources and operations and group-based serialisation that stabilises the contract with the client. The consistency of the model is automatically maintained by listeners that preserve the system's invariants. In addition, file management is robust thanks to the temporary-to-permanent workflow and the precise definition of data paths.

As for weaknesses, the technology stack is at the end of its cycle, Symfony 2.8, API, and ExtJS client, making it difficult to adopt newer practices and libraries.

The authorisation system could be improved for scenarios that require more granular permissions.

The improvements incorporated in this work directly address the lack of documentation and operational reinforcement, and pave the way for a major stack upgrade with less risk.

# 5 Granular Permission Model

## 5.1 Objectives and Context

Before this work, PartKeepr only distinguished between two profiles: normal user and administrator. With this binary scheme, anyone who accessed the application could, in practice, perform critical actions: deleting parts (and with them their stock history), creating or removing users, or emptying a warehouse by mistake directly from the grid. There were no intermediate points.

This approach clashed with basic good practices:

- Separation of duties (ISO 27001 – Annex A.9) [13]: without intermediate roles it was impossible to know who was responsible for what.
- Integrity and availability (GDPR art. 32) [14]: if anyone can alter critical data, traceability is broken.

In real environments such as laboratories, workshops, or SMEs where several people share this tool, the risk of accidental (or malicious) modifications was evident. For example, an intern adjusting stock, an operator deleting a reference "to tidy up," or a technician changing prices without control.

## 5.2 Tangible benefits after the change

The shift from the binary model to a granular permission scheme was noticeable from day one. These are the clearest effects in daily operation:

- **Accountability**. User management is reserved for administrators (admin = 1). Only administrators can create or delete users.
- **Operational scalability without giving away privileges**. A line manager can create/update items (canCreateComponents = 1) to keep the catalogue up to date, but cannot delete them (canDeleteComponents = 0). This allows work to be distributed without compromising security.
- **Cleaner interface**. Permission checkboxes and administrative options are only shown to those who can use them; others do not see buttons that do not apply to them.

Overall, the system is safer, easier to operate, and more predictable: each profile sees and can do exactly what corresponds to their role.

## 5.3 Three Before-and-After Scenarios

1. An inexperienced operator deletes 200 resistors.

- **Before**: The action was possible from the grid: immediate deletion and negative stock in seconds.
- **Now**: The 'Delete' button does not even appear for that profile. If someone forces the request, the server returns 403 and the UI displays a 'Permission Denied' warning. The inventory does not change.

2. An intern creates a test account and leaves it open.

- **Before**: They could create users and leave them active, with unnecessary privileges.
- **Now**: Account creation/deletion requires admin = 1. The intern can work with parts if their role allows it, but cannot touch administration.

3. The purchasing manager imports 50 new components.

- **Before**: indistinguishable from an administrator.
- **Now**: With canCreateComponents = 1 and canDeleteComponents = 0, they can import and create components without the ability to delete them. The flow is agile and the risk is limited.

## 5.4 Requirements and adopted design

| Functional requirement | Applied Solution | Success Indicator |
|---|---|---|
| Grant "create component" and "delete component" permissions without giving full administrator role. | Three boolean flags in PartKeeprUser: admin, canCreateComponents, canDeleteComponents. | Add Part, Delete Part, and Edit Part buttons operate only if the corresponding flag equals 1. |
| Update the UI so that only an administrator can change these flags. | Checkboxes hidden from users without admin = 1 in UserEditor.js. | Attempt by non-admin user ⇒ Access Denied message and changes not persisted. |

Binary flags were chosen instead of a full role table for two reasons:

- The original infrastructure already contained the isAdmin field as TINYINT(1), and continuity with the legacy code was required.
- The intended granularity was limited to three specific permissions, so a complete role hierarchy would have added unnecessary complexity.

## 5.5 Schema migration

During the second phase, it was necessary to update the PartKeeprUser table to store two types of data that previously did not exist:

- What each user can do (the new permissions).
- A couple of additional profile attributes (phone number and photo).

This was achieved with a single SQL script called init.sql, which is included in the partkeepr_repository. It is written in universal SQL so that it works equally well on a fresh installation or on a server that has been running for years. The trick is to always use ADD COLUMN IF NOT EXISTS: if the column already exists, the database skips it and continues.

```sql
USE partkeepr;

-- Add admin field (if it doesn't exist)
ALTER TABLE PartKeeprUser
ADD COLUMN IF NOT EXISTS admin TINYINT(1) DEFAULT 0;

-- Add permissions for components
ALTER TABLE PartKeeprUser
ADD COLUMN IF NOT EXISTS canCreateComponents TINYINT(1) DEFAULT 0,
ADD COLUMN IF NOT EXISTS canDeleteComponents TINYINT(1) DEFAULT 0;

-- Add phone number
ALTER TABLE PartKeeprUser
ADD COLUMN IF NOT EXISTS phoneNumber VARCHAR(20) NULL;

-- Add route to profile photo
ALTER TABLE PartKeeprUser
ADD COLUMN IF NOT EXISTS photo VARCHAR(255) NULL;
```

*Figure 4. Script for updating the Partkeepr schema*

## 5.6 Checkbox control

The goal of this block is to show or hide the permission fields (Admin, Can Create Components, Can Delete Components) depending on who is editing. The logic resides in two areas: the initial definition of the form and the onStartEdit hook.

In the list of items, the container with the text fields, checkboxes, etc., the three permission checkboxes are created with hidden: true. In other words, by default they are not rendered on screen.

```
    xtype: 'checkbox',
    name: 'canCreateComponents',
    fieldLabel: i18n("Can Create Components"),
    hidden: true
}, {
    xtype: 'checkbox',
    name: 'canDeleteComponents',
    fieldLabel: i18n("Can Delete Components"),
    hidden: true
}, {
    xtype: 'checkbox',
    name: 'admin',
    fieldLabel: i18n("Administrator"),
    itemId: 'adminCheckbox',
    hidden: true
}, {
```

*Figure 5. Definition of checkboxes in the user editor*

Every time the editor is opened, the onStartEdit handler is registered (with a small delay to ensure that the form has been rendered). This handler decides the visibility and state of the sensitive fields.

```
this.on("startEdit", this.onStartEdit, this, { delay: 200 });
this.callParent();
```

*Figure 6. Registration of the start edit hook (startEdit → onStartEdit)*

Steps:

- The logged-in user is retrieved and the isAdmin flag is calculated by reading its admin attribute.

```
onStartEdit: function () {
    const editor = this;
    const currentUser = PartKeepr.getApplication().getLoginManager().getUser();
    const isAdmin = currentUser && currentUser.get("admin") === true;
```

*Figure 7. Detection of administrator user inside onStartEdit()*

- Show the checkboxes only to administrators: this.down() searches for the control by name or itemId and calls show(). If isAdmin is false, the block is not executed and the controls remain invisible.

```
if (isAdmin) {
    this.down("field[name=canCreateComponents]").show();
    this.down("field[name=canDeleteComponents]").show();
    this.down("#adminCheckbox").show();
}
```

*Figure 8. Logic in UserEditor.js to show permission checkboxes only to administrators*

## 5.7 Practical Advantages

- **Error prevention**. An operator without privileges cannot self-assign permissions because they do not even see the fields.
- **Clean experience**. Administrators can edit everything in a single screen, without having to open a separate form.
- **Maintainability**. If another flag is added tomorrow, it is enough to create it hidden and add a show() call in the same block.

Overall, this small portion of code links the server-side authorisation logic with client usability, reinforcing consistency and security without adding unnecessary complexity.

## 5.8 Unified permission reading

In resources/public/js/partkeepr.js lies the first point touched by the browser when PartKeepr starts. There, the three new indicators admin, canCreateComponents, and canDeleteComponents have been added, so that they are loaded only once, at the same moment the user connects.

### Reception of permissions

As soon as the credential is validated, the LoginManager returns the User model via the server normaliser. In the onLogin() handler I store in memory (application global state) whether the session belongs to an administrator.

```
this.setAdmin(this.getLoginManager().getUser().get("admin"));
```

*Figure 9. Marking the user as administrator at startup onLogin()*

This memory record persists until logout and avoids having to query the backend again in each view.

### Quick access to permissions with getSecurityContext()

For parts of PartKeepr that only need to know whether they can create or delete, the following was added:

```
getSecurityContext: function () {
    return {
        canCreateComponents: this.getLoginManager().getUser().get("canCreateComponents"),
        canDeleteComponents: this.getLoginManager().getUser().get("canDeleteComponents")
    };
}
```

*Figure 10. Centralised access to canCreateComponents and canDeleteComponents*

Thanks to this, for example, a button can be disabled very easily.

All of this means that permissions are checked only once, resulting in less network traffic and less duplicated logic.

## 5.9 Permissions in part management

In daily inventory management, there are four particularly sensitive actions: adding, deleting, and duplicating parts. Before this improvement, the flow went directly to the database. Now, all these actions first query the user's security context and only continue if the appropriate permission exists. The idea is simple and effective: check at the beginning and abort with a clear message if not allowed.

For the Add button function, a check for canCreateComponents has been added. If the user does not have the necessary permission, an alert message is triggered.

```
onItemAdd: function (defaults)
{
    if (!PartKeepr.getApplication().getUser().get("canCreateComponents")) {
        Ext.Msg.alert(i18n("Permission Denied"), i18n("You don't have permission to create components."));
        return;
    }
    var j = Ext.create("PartKeepr.PartEditorWindow", {
        partMode: 'create'
    });

    var defaultPartUnit = PartKeepr.getApplication().getPartUnitStore().findRecord("default", true);

    Ext.apply(defaults, {});

    var record = Ext.create("PartKeepr.PartBundle.Entity.Part", defaults);

    if (this.getSelectedCategory() !== null)
    {
        record.setCategory(this.getSelectedCategory());
    } else
    {
        record.setCategory(this.tree.getRootNode().firstChild);
    }

    record.setPartUnit(defaultPartUnit);

    j.editor.editItem(record);
    j.editor.on("partSaved", this.onNewPartSaved, this);
    j.show();

    return j;
},
```

*Figure 11. Checking canCreateComponents when adding a new component*

In the onItemDelete method, if the user has permission to delete a component, the system asks whether they really want to delete it, and if the answer is affirmative, deletePart is called. Otherwise, an alert message is shown.

```
/**
 * Called when the delete button was clicked.
 *
 * Prompts the user if he really wishes to delete the part. If yes, it calls deletePart.
 */
onItemDelete: function ()
{
    if (!PartKeepr.getApplication().getUser().get("canDeleteComponents")) {
        Ext.Msg.alert(i18n("Permission Denied"), i18n("You don't have permission to delete components."));
        return;
    }
    var r = this.grid.getSelectionModel().getLastSelected();

    Ext.Msg.confirm(i18n("Delete Part"), sprintf(i18n("Do you really wish to delete the part %s?"), r.get("name")),
        this.deletePart, this);
},
```

*Figure 12.Verification of canDeleteComponents and confirmation before Delete*

For the onDuplicateItemWithBasicData and onDuplicateItemWithAllData methods, the same logic is applied.

```
/**
 * Creates a duplicate with the basic data only from the selected item. Loads the selected part and calls
 * createPartDuplicate after the part was loaded.onDuplicateItemWithBasicData
 */
onDuplicateItemWithBasicData: function ()
{
    if (!PartKeepr.getApplication().getUser().get("canCreateComponents")) {
        Ext.Msg.alert(i18n("Permission Denied"), i18n("You don't have permission to duplicate components."));
        return;
    }

    const r = this.grid.getSelectionModel().getLastSelected();
    this.loadPart(r.getId(), Ext.bind(this.createPartDuplicate, this));
},
```

*Figure 13. Permission control when duplicating with basic data*

```
/**
 * Creates a full duplicate from the selected item. Loads the selected part and calls createFullPartDuplicate
 * after the part was loaded.
 */
onDuplicateItemWithAllData: function ()
{
    if (!PartKeepr.getApplication().getUser().get("canCreateComponents")) {
        Ext.Msg.alert(i18n("Permission Denied"), i18n("You don't have permission to duplicate components."));
        return;
    }

    const r = this.grid.getSelectionModel().getLastSelected();
    this.loadPart(r.getId(), Ext.bind(this.createFullPartDuplicate, this));
},
```

*Figure 14. Permission control when duplicating with all data*

These functions are important because they refer to the principle of defence in depth [15], where the first barrier (disabled buttons) lives in the grid, and the second barrier resides in the controller. There is also cohesion, since validations are located at the beginning of the method and not mixed in.

## 5.10 Intelligent visibility of permission checkboxes

The UserEditor now decides in real time which checkboxes should be shown depending on the role of the user editing.

The logic resides at the end of onStartEdit() and can be summarised in three steps.

It calculates once whether the logged-in person has admin = true. This avoids repeating checks on every click and keeps the method readable.

```
onStartEdit: function () {
    const editor = this;
    const currentUser = PartKeepr.getApplication().getLoginManager().getUser();
    const isAdmin = currentUser && currentUser.get("admin") === true;
```

*Figure 15. Calculation of role (isAdmin) when opening the user editor*

With the following code, the three sensitive fields are displayed. The checkboxes only appear when the editor is opened by an administrator. For a normal user, the form looks cleaner and does not invite tampering.

```
if (isAdmin) {
    this.down("field[name=canCreateComponents]").show();
    this.down("field[name=canDeleteComponents]").show();
    this.down("#adminCheckbox").show();
}
```

*Figure 16. Conditional presentation: show permission checkboxes only to administrators*

## 5.11 Permission visualisation in the interface, before and after

In the screenshot of the original version, the user editor showed the account name, email, password, and as the only status indicator, the Active checkbox. Nothing more—no trace of privileges or any information to help identify the person behind the account.

With that approach, any user accessing the editor could, without restrictions, create or delete components and even register or remove other users. The risk of accidental, or malicious, deletion was real.

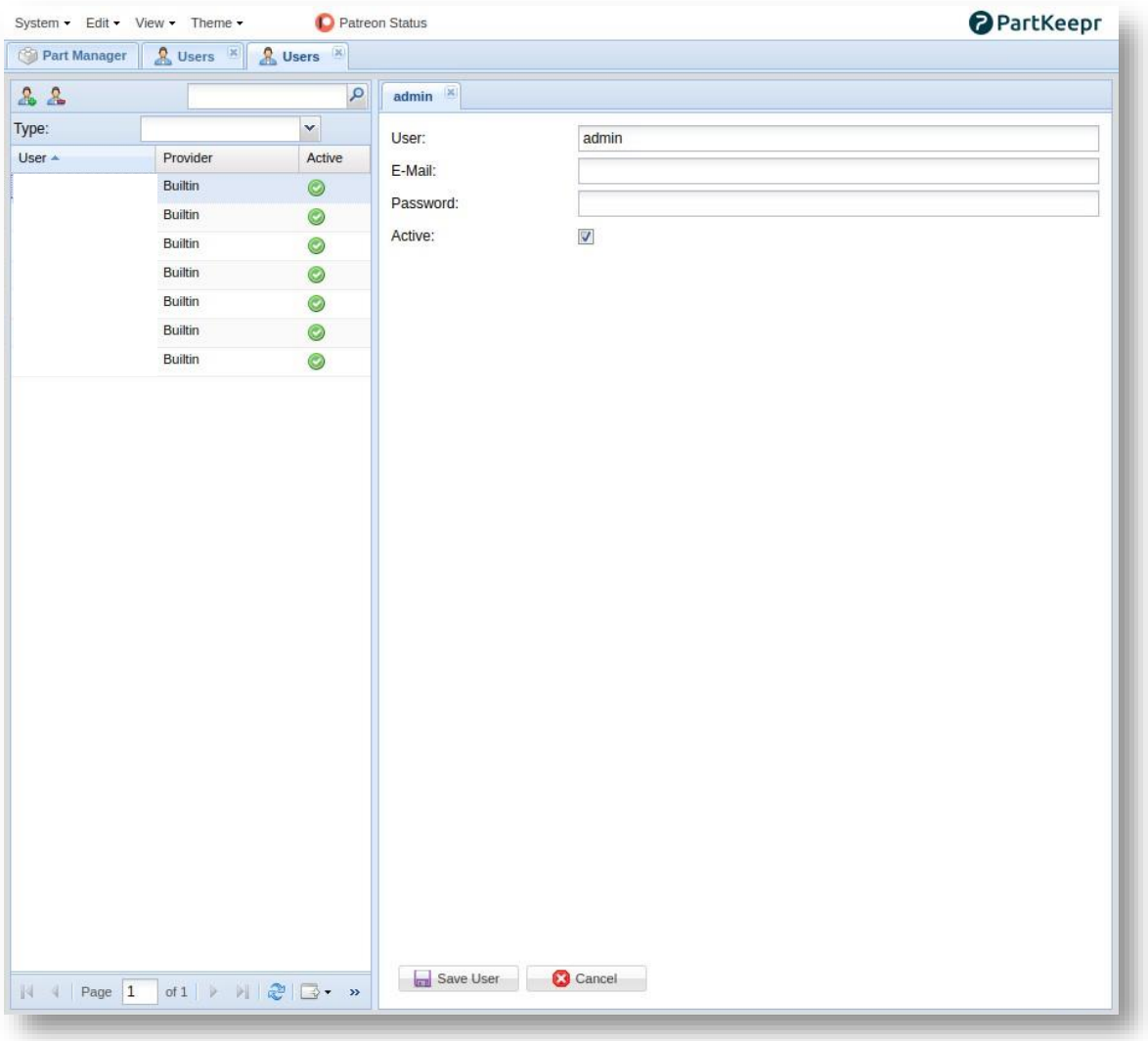



*Figure 17. Original view of the user editor*

The current screenshot demonstrates the qualitative leap. Below the basic fields, three new checkboxes appear: Can Create Components, Can Delete Components, and Administrator.

They are only rendered when the editor is opened by an administrator, so an operator without privileges does not even know they exist, much less activate them. Alongside these controls, the user's phone number and photo have been added two details that, in the daily life of the laboratory, facilitate quick identification and internal communication. With this change, the editor ceases to be a simple form and becomes the entry point to PartKeepr's security model. The principle of least privilege is applied, roles are documented, and trust in the platform is reinforced.

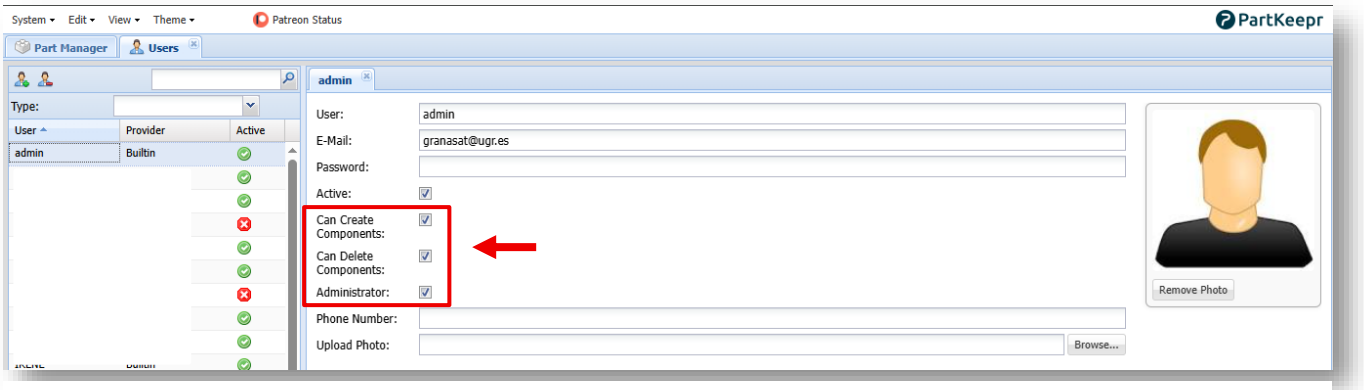



*Figure 18. Updated view of the user editor when the user has administrator permissions*

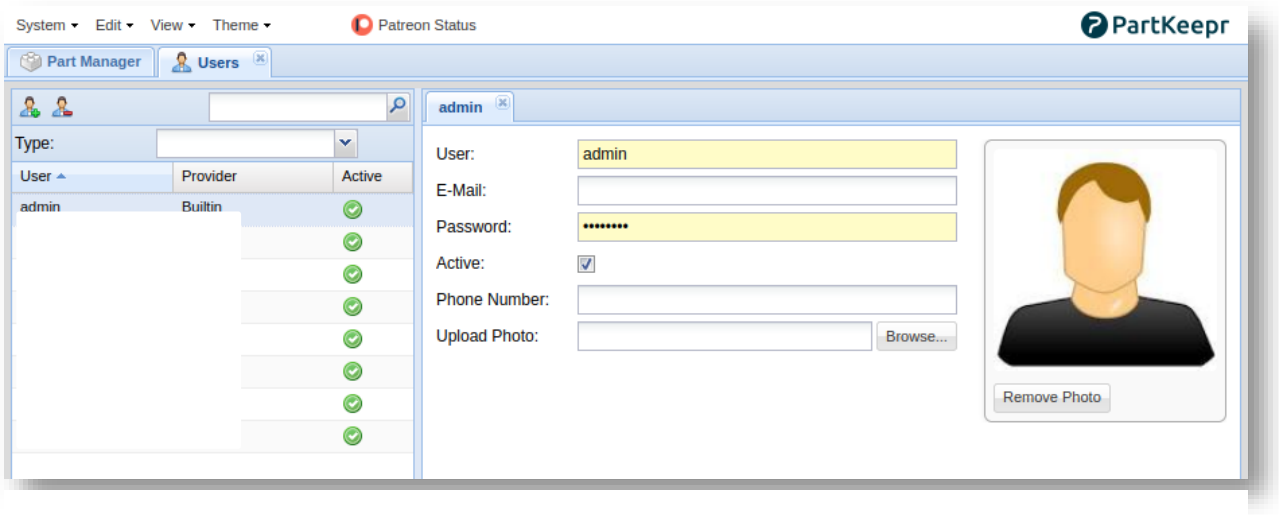



*Figure 19. Updated view of the user editor when the user does NOT have administrator permissions*

## 5.12 Lessons learned and future extensions

The move from an all-or-nothing model to a granular permission scheme left several clear conclusions: the simpler and more explicit the control, the fewer operational errors occur; and yet, security must still be verified in two layers (interface and server). From here, lines of improvement remain open that allow growth without reworking what has already been implemented.

**Keep it simple**

We chose three very straightforward indicators, admin, canCreateComponents and canDeleteComponents, because they addressed everyday use cases without rewriting PartKeepr. The experience confirms that less is more: rules understandable by everyone, short documentation, and frictionless deployment. If new needs appear in the future (e.g. a Purchasing role that can view prices but not edit them), the current design already paves the way: it would suffice to introduce a role table and map it to specific permissions without breaking compatibility.

**Layer of defense**

Validation occurs twice: first in the UI (so that the user does not even see what they cannot use), and then in the API (so that a manual attempt cannot slip through). This double lock adds very little code and avoids costly incidents. [16]

**Show only what is necessary**

Hiding controls from those who cannot use them clears the screen and reduces clicks. The user editor shows permission checkboxes only to administrators; for the rest, the interface focuses on actual work and lowers the risk of accidental changes.

# 6 Enrichment of user profiles

## 6.1 Objective and context

In the original version, users were more like accounts than people, since they only had fields for name, email, password, and little else. In workshops and laboratories, this was insufficient. Two needs came up repeatedly:

1. The ability to locate someone quickly via phone.
2. The ability to visually identify the user with a photo.

From these needs came this improvement which added **phoneNumber** and **photo** to the **profile** and ensured the interface used them naturally.

## 6.2 What was done

- **New fields in the user entity**: phoneNumber (string, 20) and photo (string, 255).

```
/**
 * @ORM\Column(type="string", length=20, nullable=true)
 * @Groups({"default"})
 */
private $phoneNumber;

/**
 * @ORM\Column(type="string", length=255, nullable=true)
 * @Groups({"default"})
 */
private $photo;
```

*Figure 20. New fields in the User entity: phoneNumber and photo*

- **Getters.** For the phone number the stored value is returned. For the photo the stored image is returned or, if no user photo exists, a default image.

```php
public function getPhoneNumber()
{
    return $this->phoneNumber;
}

public function getPhoto()
{
    return $this->photo ?: '/images/user_whitout_foto.jpg';
}
```

*Figure 21. Profile getters returning the phone number and getPhoto() with default image when no photo is uploaded*

- **Setters.** The phone number is saved to the object and for the photo the path or URL is saved to the object.

```php
public function setPhoneNumber($phoneNumber)
{
    $this->phoneNumber = $phoneNumber;
}

public function setPhoto($photo)
{
    $this->photo = $photo;
}
```

*Figure 22. Profile setters persistence of phone and path or URL of the user's photo*

- **Photo** upload from the UI with local file selection and a remove photo button.

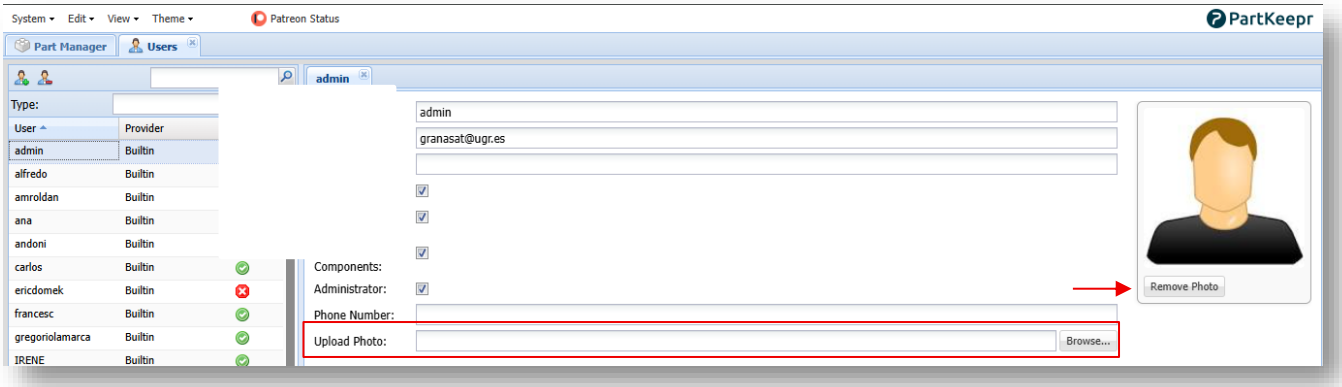



*Figure 23. Photo upload from the interface file selector and Remove Photo button*

- **Backend** action to persist the file, resize it to 120×160 and return the public path.
- **Routing and container** permissions to serve /uploads/users/...
- **Schema and deployment**. Database update with Doctrine and volume mapping in Docker to ensure persistence.

## 6.3 Requirements and adopted design

The goal was straightforward. Each user should be able to add a contact phone number and a profile photo and this should be displayed naturally in the interface. The design relied on minimal changes to data, API and UI without touching sensitive core components.

Profiles were extended with two optional fields in the User entity (src/PartKeepr/AuthBundle/Entity/User.php)

- **phoneNumber** (string, length 20, nullable=true) and photo (string, length 255, nullable=true).

  Both are annotated with @Groups({"default"}) so they are exposed via the API without extra steps. The getPhoto() getter returns a fallback path (images/usuario_sin_foto.jpg) when no image is stored so the interface always has something to display.

The schema was updated with

- php app/console doctrine:schema:update --force.

In Docker environments, volumes for src, web and app were mounted so the container would pick up code changes and the schema update would not fail with “Unknown column”. This kept the database aligned without complex migrations.

In the **interface**, the user editor (UserEditor.js) was updated with:

- A text field for the phone number.
- A filefield for photo upload. Two practical details are worth noting. The size is limited to 2 MB in the field’s own change handler. When the user selects an image an Ext.Ajax.request is sent to /api/upload_photo with a FormData. If a photo already existed its path (currentPhoto) is also sent so the server can delete the previous one.

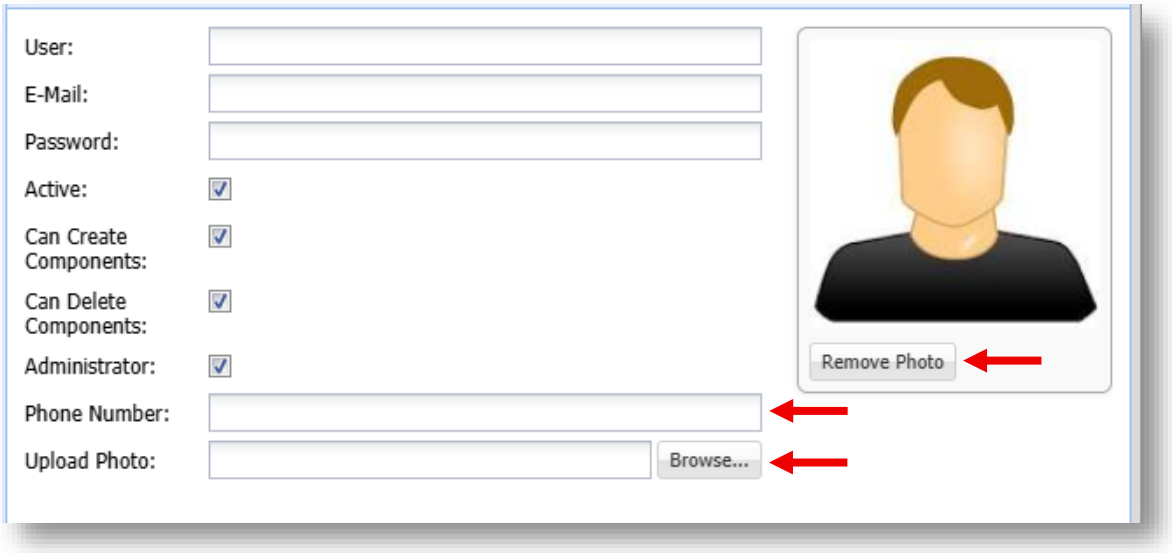



*Figure 24. Updated and complete User Editor form*

In the **backend**, UploadUserPhotoAction saves the file in web/uploads/users, ensures the directory exists, generates a unique name and resizes the image to 120×160 with Imagick. It then returns a JSON with the public path /uploads/users/....

To enable this, a new routing file (routing_upload_user_photo.yml) was created and registered in app/config/routing.yml and the cache was cleared afterwards. Permissions were also prepared for the web/uploads directory to allow write access by the web process.

```php
class UploadUserPhotoAction
{
    public function __invoke(Request $request)
    {
        /** @var UploadedFile|null $file */
        $file = $request->files->get('file');

        if (!$file) {
            throw new HttpException(400, "No file uploaded");
        }

        $allowedMimeTypes = ['image/jpeg', 'image/png'];
        $mimeType = $file->getMimeType();

        if (!in_array($mimeType, $allowedMimeTypes)) {
            throw new HttpException(400, "Unsupported file type: $mimeType. Only JPG and PNG are allowed.");
        }

        $uploadDir = __DIR__ . '/../../../web/uploads/users';
        $filesystem = new Filesystem();
        $filesystem->mkdir($uploadDir);

        // Delete the previous one if 'currentPhoto' is provided (from the client)
        $currentPhoto = $request->get('currentPhoto');
        if ($currentPhoto) {
            $oldFilePath = $uploadDir . '/' . basename($currentPhoto);
            if ($filesystem->exists($oldFilePath)) {
                $filesystem->remove($oldFilePath);
            }
        }

        $extension = $file->guessExtension();
        $filename = uniqid() . '.' . $extension;
        $path = $uploadDir . '/' . $filename;

        // Resize to 200x250 (ID card format)
        $srcImage = ($mimeType === 'image/jpeg')
            ? imagecreatefromjpeg($file->getPathname())
            : imagecreatefrompng($file->getPathname());

        $targetWidth = 200;
        $targetHeight = 250;

        $resizedImage = imagecreatetruecolor($targetWidth, $targetHeight);
        imagecopyresampled(
            $resizedImage,
            $srcImage,
            0,
            0,
            0,
            0,
            $targetWidth,
            $targetHeight,
            imagesx($srcImage),
            imagesy($srcImage)
        );

        if ($mimeType === 'image/jpeg') {
            imagejpeg($resizedImage, $path);
        } else {
            imagepng($resizedImage, $path);
        }

        imagedestroy($resizedImage);
        imagedestroy($srcImage);

        return new JsonResponse(['photo' => '/uploads/users/' . $filename]);
    }
}
```

*Figure 25. Backend action UploadUserPhotoAction*

From the user's perspective, the editor shows a thumbnail in a side panel. If the upload succeeds it updates instantly with the new URL. If no photo exists the default image is shown automatically. Everything happens within the same form without reloading the page or requiring additional confirmation.

## 6.4 Test and minor issues

- **The router did not detect the new route**. A specific YAML file was created and cache cleared, verified with debug:router.
- **File format**. Some .jpg images failed due to fileinfo. The extension was enabled and extension or MIME checking reinforced.
- **Persistence in Docker**. Without volumes all data was lost after restart. This was solved by mapping src, web and app as described above**.**

## 6.5 Result

Profiles are no longer just username and email. With phone and photo, PartKeepr becomes more personal and operational without adding complexity. A schema command, a new route and a thumbnail on screen are enough. Everything is integrated with the rest of the system and ready to grow when needed.

# 7 Thumbnails in attachments

## 7.1 Problem addressed

In the attachments grid, images (JPG, PNG, etc.) were displayed the same way as any other file, only with a text name. To quickly review visual documentation, it was necessary to open each file. The goal was for images to appear as thumbnails directly in the table while keeping the previous behaviour for non-graphic files (PDF, ZIP and so on).

### Before

In the attachments tab of the part editor, the grid only displayed the file name and size.

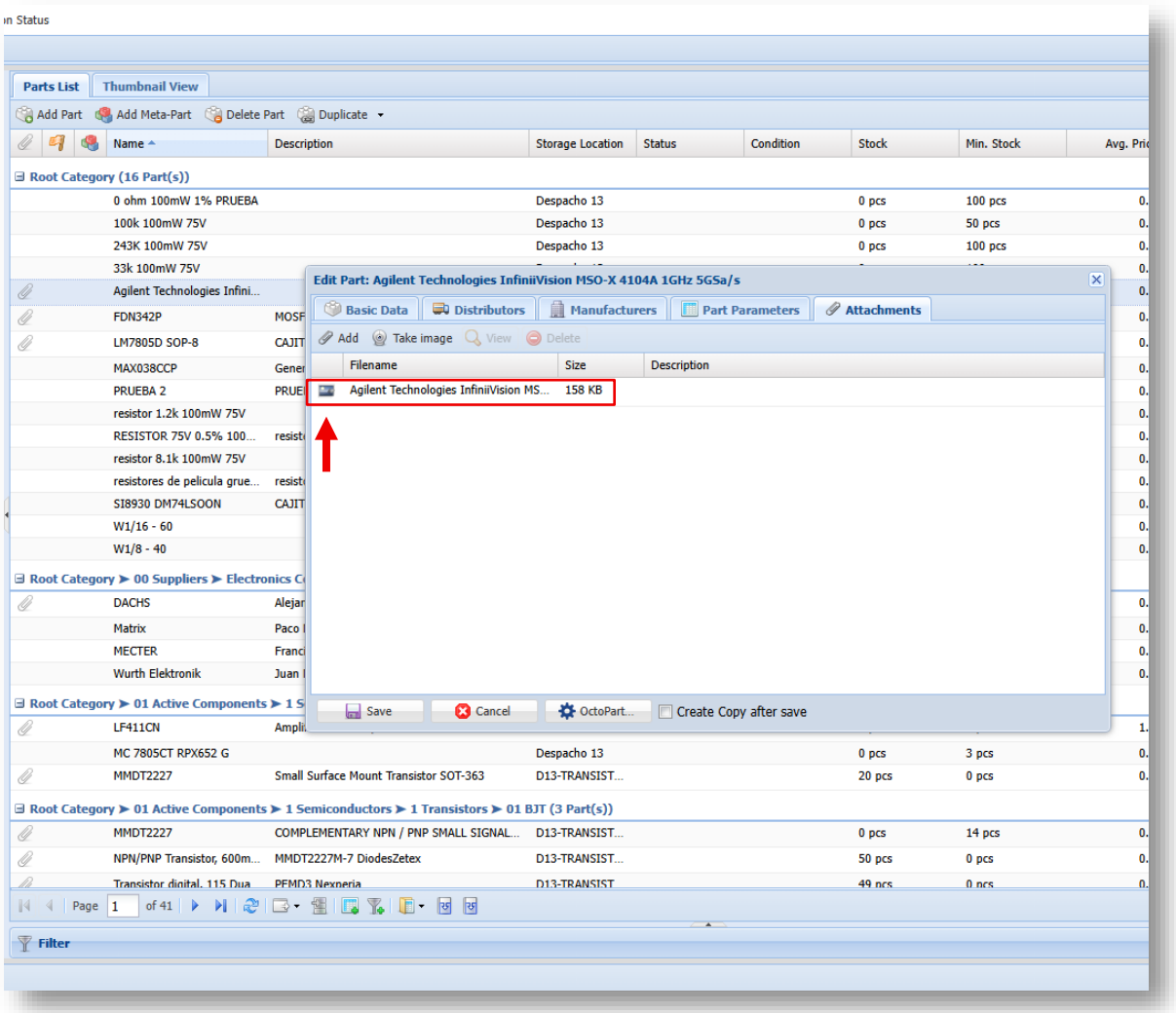



*Figure 26. Previous view of the interface in the Attachments tab*

**After**

In the same tab, the grid now renders a thumbnail of the attached file.

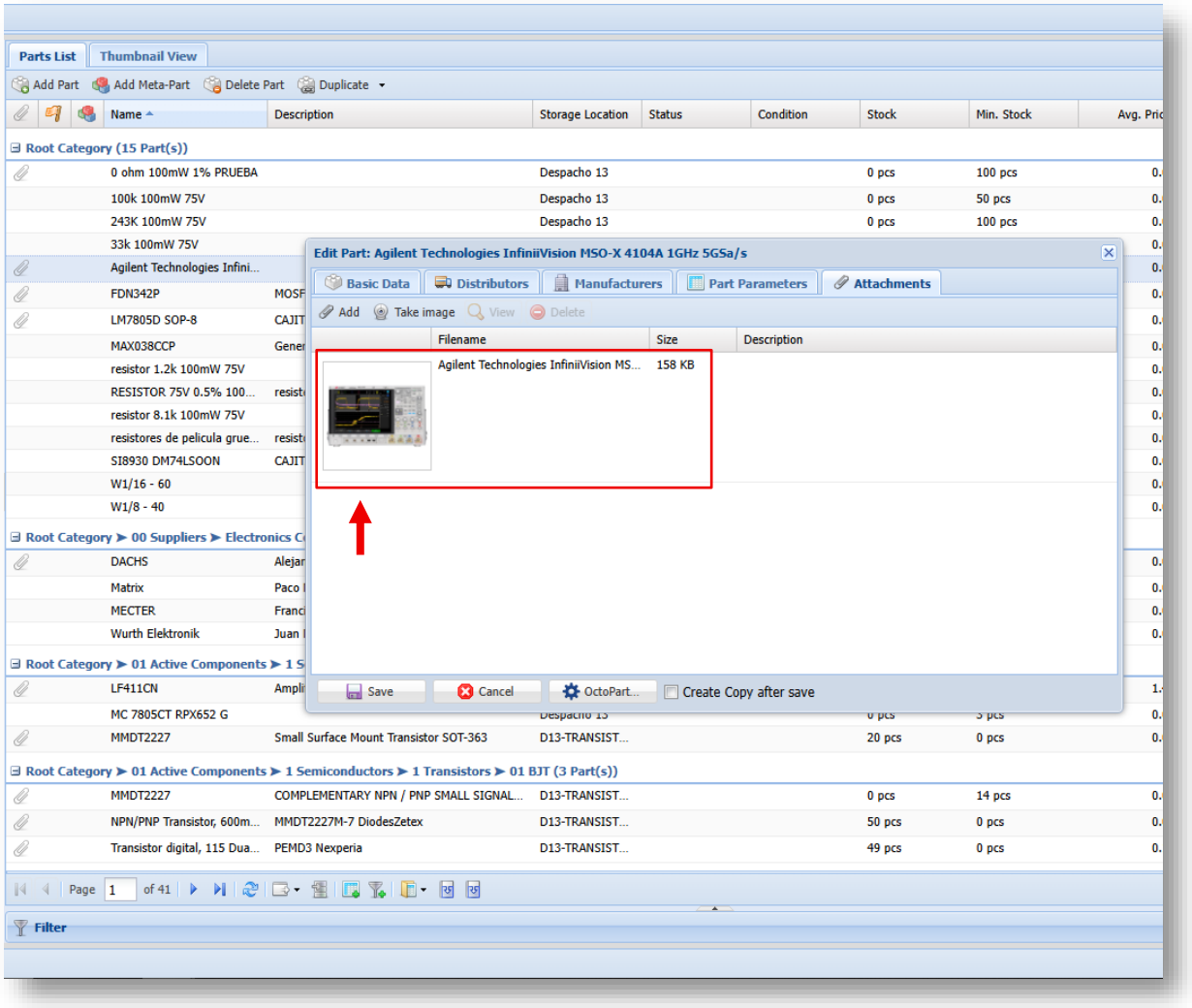



*Figure 27. Updated view of the interface in the Attachments tab*

## 7.2 What has changed and how it works

The renderer of the column that displays the file was customised.

```
this.columns = [
    {
        text: '',
        width: 110,
        renderer: function (value, metadata, record) {
            const filename = record.get('originalFilename') || '';
            const isImage = /\.(jpe?g|png|gif|bmp)$/i.test(filename);
            const id = record.get('@id');
            if (isImage && id) {
                return '<div style="padding: 4px"><img src="' + id + '/getFile" width="100" height="100" ' +
                       'style="object-fit: contain; border: 1px solid #ccc; cursor: pointer" ' +
                       'onclick="window.open(\'' + id + '/getFile\', \"_blank\")"></div>';
            } else {
                return '<img src="' + record.getId() + '/getMimeTypeIcon" width="24" height="24">';
            }
        }
    },
```

*Figure 28. Code of the updated Attachments column*

1. **Image detection:** isImage uses a regular expression against the extension of originalFilename to decide if it is jpg, jpeg, png, gif or bmp.
2. **URL construction:** with record.get('@id') (the JSON-LD path of the resource) it builds id + '/getFile', which is the standard PartKeepr endpoint for serving the file.
3. **Styled thumbnail:** the <img> is forced to 100×100 px with object-fit: contain and a subtle border so that any proportion fits without distortion.
4. **Direct action:** onclick opens the attachment in a new tab when the thumbnail is clicked.
5. **Elegant fallback:** if the attachment is not an image, a MIME icon is shown via …/getMimeTypeIcon.

## 7.3 Why this solution is sufficient and safe

There was no need to change the database or the API. Existing endpoints (…/getFile and …/getMimeTypeIcon) were reused. This covers any image served by PartKeepr (JPEG, PNG, GIF, BMP) and for other files everything continues working as before. The change lives only in the presentation layer, so the models and server logic remain intact.

From a security perspective everything stays within the standard framework. The …/getFile URLs go through the usual authentication and authorisation. No system paths are exposed, only resource IDs. The renderer does not inject user data into sensitive attributes, the only interpolation is the URL generated by the backend.

A note for the future. At present the browser reduces the image to thumbnail size (100×100). If very large files begin to be used, a thumbnail endpoint could be added (for example …/getFile?maxWidth=100&maxHeight=100) so that the small version is served directly and traffic is saved.

## 7.4 Observable benefits

Attachments can be identified at a glance without opening anything, resulting in fewer clicks and less wasted time. The interface becomes more consistent with other areas of PartKeepr that already display thumbnails. Fewer mistakes occur when linking images to parts because it is clear whether the file is correct. And if more detail is needed, a double click opens the file in another tab with a larger and clearer view.

# 8 Nexar Integration (replacement of the former Octopart)

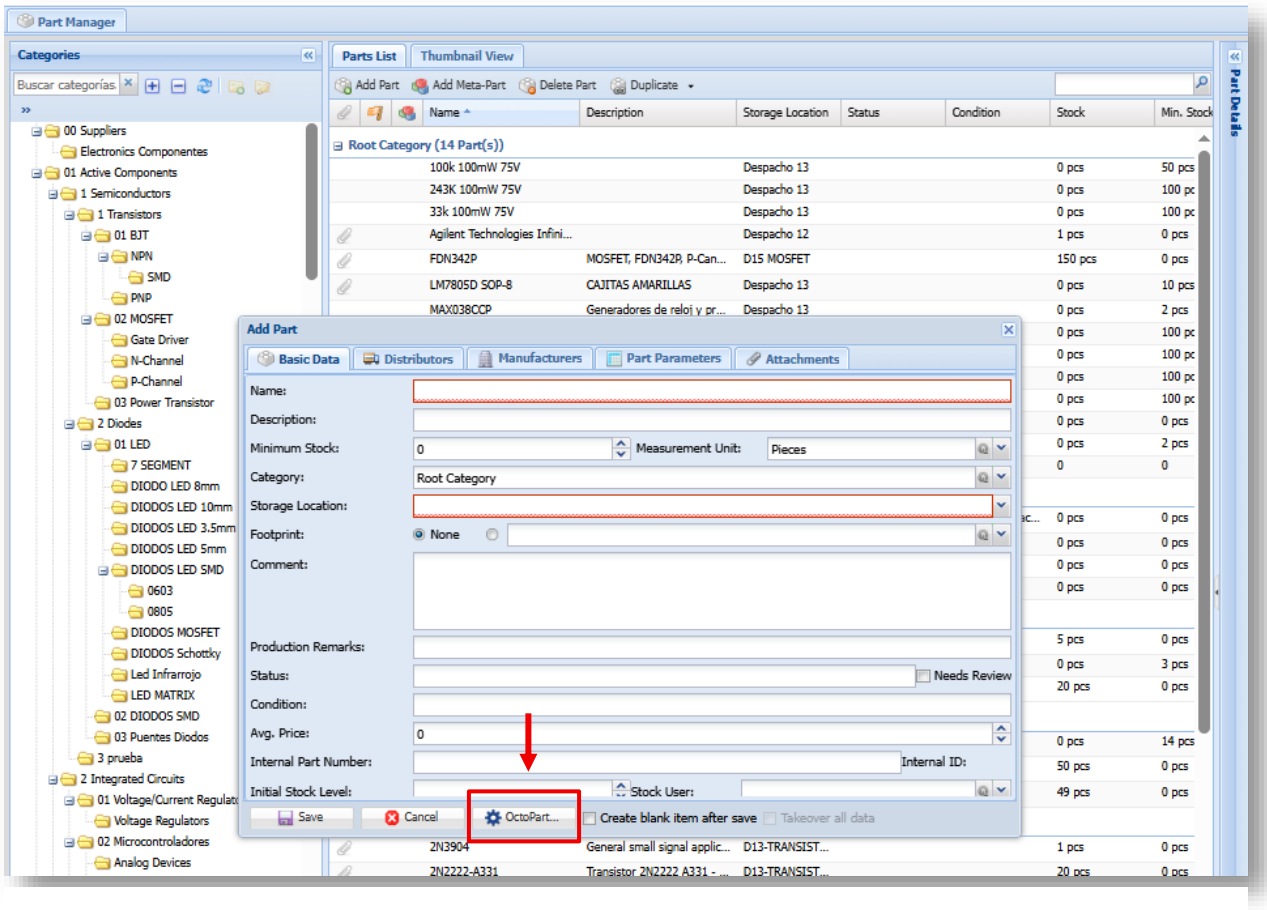



*Figure 29. UI window with Octopart button*

For years PartKeepr included a connector to Octopart based on its API v3. That API was discontinued when Octopart became part of Nexar (Altium), and from that moment the Octopart button stopped returning useful results. In addition, the project code expected REST responses from the old model, which was incompatible with the modern GraphQL scheme used by Nexar. The symptom for the user was clear. Searching for an MPN resulted in an error. [17]

The objective of the change was simple to explain but demanding internally. The aim was to migrate the search and import of components to Nexar GraphQL while maintaining the user experience of searching by MPN and retrieving manufacturer, distributors, prices, images and datasheets without touching the database.

## 8.1 What was changed

The core of the work lies in the service located at OctoPartBundle/Services/OctopartService.php. The service now authenticates with OAuth2 against https://identity.nexar.com/connect/token and stores the access token with its expiry time in memory, leaving a small safety margin before expiration.

With the token in hand, it communicates with https://api.nexar.com/graphql and launches GraphQL queries such as supSearchMpn. [18]

```
    public function __construct()
    {
        $this->searchQuery = <<<GRAPHQL
query SearchParts(\$q: String!) {
  supSearchMpn(q: \$q, country: "ES") {
    hits
    results {
      description
      part {
        id
        name
        mpn
        totalAvail
        images {
          url
        }
        documentCollections {
          documents {
            name
            url
            mimeType
          }
        }
        cad {
          downloadUrlAltium
          downloadUrlEagle
          downloadUrlOrcad
          downloadUrlKicad
        }
        sellers {
          company {
            name
          }
          offers {
            prices {
              quantity
              price
            }
          }
        }
      }
    }
  }
}
GRAPHQL;
    }
```

*Figure 30. GraphQL query supSearchMpn code in Nexar*

From there, the relevant data is obtained including images, distributor, MPN, description, quantity, price, document collections such as datasheets and CAD models, which are then used by the controller.

To refine the results, country: "ES" is set, which can be parameterised if required by the installation.

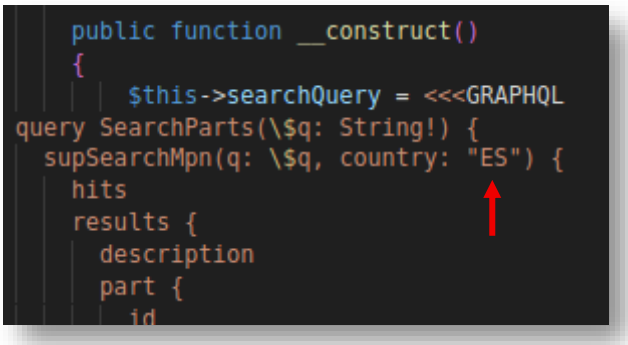



*Figure 31. Parameter setting to refine results by region*

The basic search by MPN is performed with searchPart($mpn). Its purpose is to fetch raw results directly from Nexar. It calls the authentication function Authenticate() to ensure the token is valid, then builds the GraphQL request body. If there are no GraphQL errors, the data is returned.

```php
public function searchPart($mpn)
{
    $this->authenticate();

    $query = [
        'query' => $this->searchQuery,
        'variables' => [
            'q' => $mpn
        ]
    ];

    $ch = curl_init('https://api.nexar.com/graphql');
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($ch, CURLOPT_POST, true);
    curl_setopt($ch, CURLOPT_HTTPHEADER, [
        'Content-Type: application/json',
        'Authorization: Bearer ' . $this->accessToken
    ]);
    curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($query));

    $response = curl_exec($ch);
    if (curl_errno($ch)) {
        throw new \RuntimeException('Request Error: ' . curl_error($ch));
    }

    $httpStatus = curl_getinfo($ch, CURLINFO_HTTP_CODE);
    if ($httpStatus !== 200) {
        throw new \RuntimeException('Nexar API Error: HTTP ' . $httpStatus . ' - ' . $response);
    }

    curl_close($ch);

    $data = json_decode($response, true);
    if (isset($data['errors'])) {
        throw new \RuntimeException('Nexar GraphQL Error: ' . json_encode($data['errors']));
    }

    return $data['data']['supSearchMpn']['results'] ?? [];
}
```

*Figure 32. Basic search function in Nexar*

To provide a paginated list of results, the service getPartsByQuery($query, $page) is used to paginate on the server side with 25 results per page.

```php
public function getPartsByQuery($query, $page = 1)
{
    $results = $this->searchPart($query);

    $pageSize = 25;
    $offset = ($page - 1) * $pageSize;
    $paginatedResults = array_slice($results, $offset, $pageSize);

    return [
        'hits' => count($results),
        'results' => $paginatedResults
    ];
}
```

*Figure 33. Paginated search function in Nexar*

The function getPartByMpn($mpn) retrieves the full record of a specific component with all the extra fields required to populate the part entry.

```
    public function getPartByMpn($mpn)
{
    $this->authenticate();

    $gql = <<<GRAPHQL
query SearchPart(\$mpn: String!) {
  supSearchMpn(q: \$mpn, country: "ES") {
    results {
      part {
        id
        name
        mpn
        sellers {
          company {
            name
          }
          offers {
            prices {
              quantity
              price
            }
          }
        }
        shortDescription
        category {
          name
        }
        totalAvail
        medianPrice1000 {
          quantity
          currency
          price
        }
        images {
          url
        }
        documentCollections {
          documents {
            name
            url
            mimeType
          }
        }
        cad {
          downloadUrlAltium
          downloadUrlEagle
          downloadUrlOrcad
          downloadUrlKicad
        }
        manufacturer {
          name
        }
      }
    }
  }
}
GRAPHQL;
```

*Figure 34. GraphQL query in Nexar for a complete component record*

In the HTTP controller located in OctoPartBundle/Controller/DefaultController.php, two routes are exposed:

1. **GET /api/octopart/query/** is used to fetch the list of results. The controller validates q, calls the service and for each result returned by Nexar builds a row in the grid with image, distributor, MPN, description, quantity, minimum price, datasheet and CAD models.

```php
class DefaultController extends FOSRestController
{
    /**
     * @Routing\Route("/api/octopart/query/", defaults={"method" = "GET", "_format" = "json"})
     * @Routing\Method({"GET"})
     * @View()
     */
    public function getPartsByQueryAction(Request $request)
    {
        $page = $request->query->getInt("page", 1);
        $query = $request->query->get("q", '');

        if (empty($query)) {
            throw $this->createNotFoundException("Query parameter 'q' is required.");
        }

        $service = $this->get("partkeepr.octopart_service");
        $data = $service->getPartsByQuery($query, $page);

        $responseData = [
            "hits" => $data["hits"] ?? 0,
            "results" => []
        ];

        foreach ($data["results"] as $result) {
            $part = $result["part"];
            $description = $result["description"] ?? '';

            $imageUrl = $part["images"][0]["url"] ?? '';

            $datasheetUrl = '';
            if (isset($part["documentCollections"])) {
                foreach ($part["documentCollections"] as $collection) {
                    if (isset($collection["documents"])) {
                        foreach ($collection["documents"] as $doc) {
                            if (!isset($doc["url"]) || !isset($doc["mimeType"])) {
                                continue;
                            }
                            $mimeType = strtolower($doc["mimeType"]);

                            if (empty($datasheetUrl) && $mimeType === 'application/pdf') {
                                $datasheetUrl = $doc["url"];
                                break 2;
                            }
                        }
                    }
                }
            }

            $cadModelUrl = '';
            if (isset($part["cad"])) {
                $cadFields = [
                    "downloadUrlAltium",
                    "downloadUrlEagle",
                    "downloadUrlOrcad",
                    "downloadUrlKicad"
                ];

                foreach ($cadFields as $field) {
                    if (!empty($part["cad"][$field])) {
                        $cadModelUrl = $part["cad"][$field];
                        break;
                    }
                }
```

*Figure 35. HTTP controller for the lightweight component view*

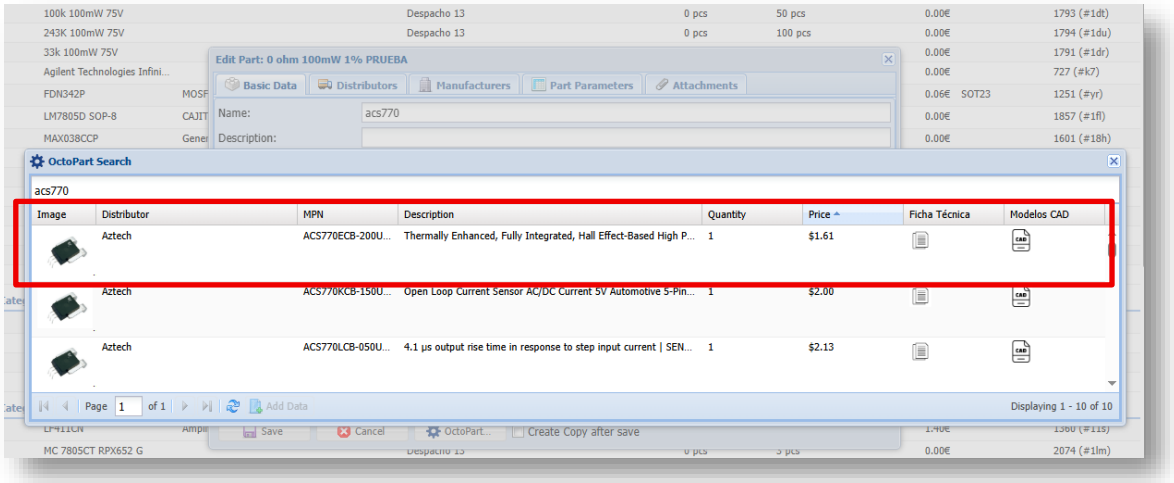



*Figure 36. UI window with component search results*

2. **GET /api/octopart/get/{mpn}** is used to select a result. When a result from the previous list is selected, the complete record is returned already normalised so that the PartKeepr editor can store all the information.

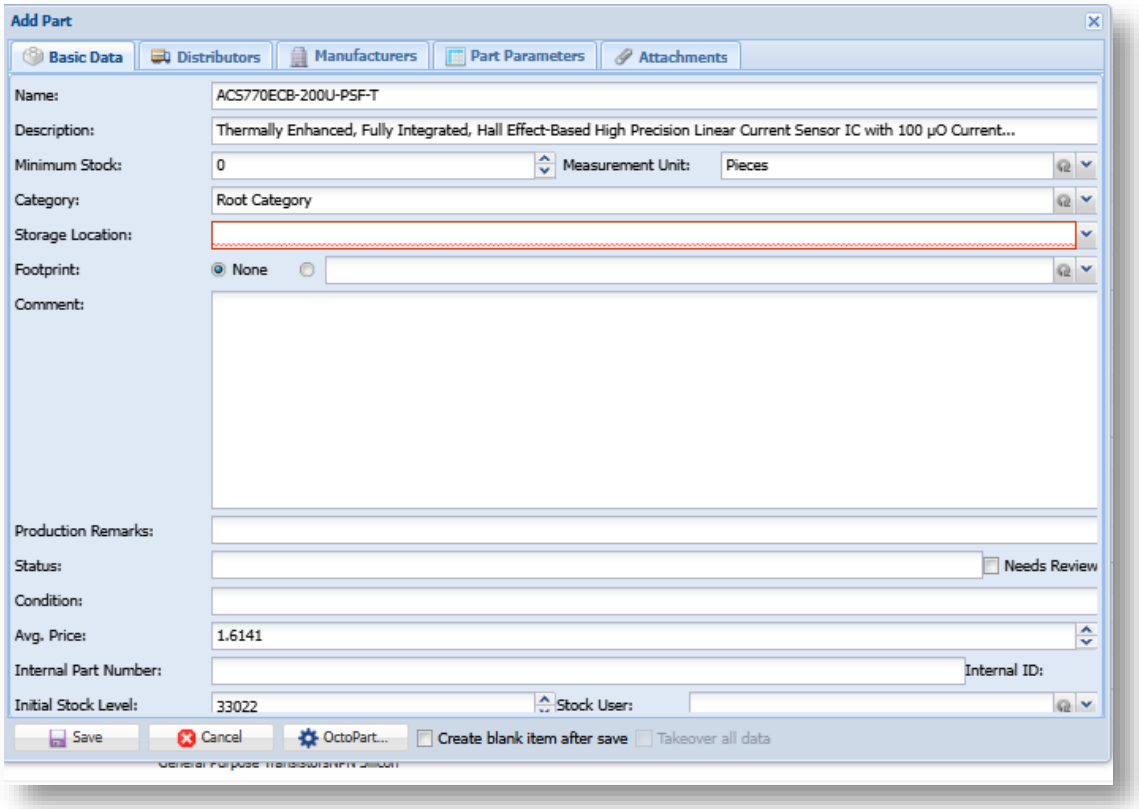



*Figure 37. UI window with basic data of the selected result*

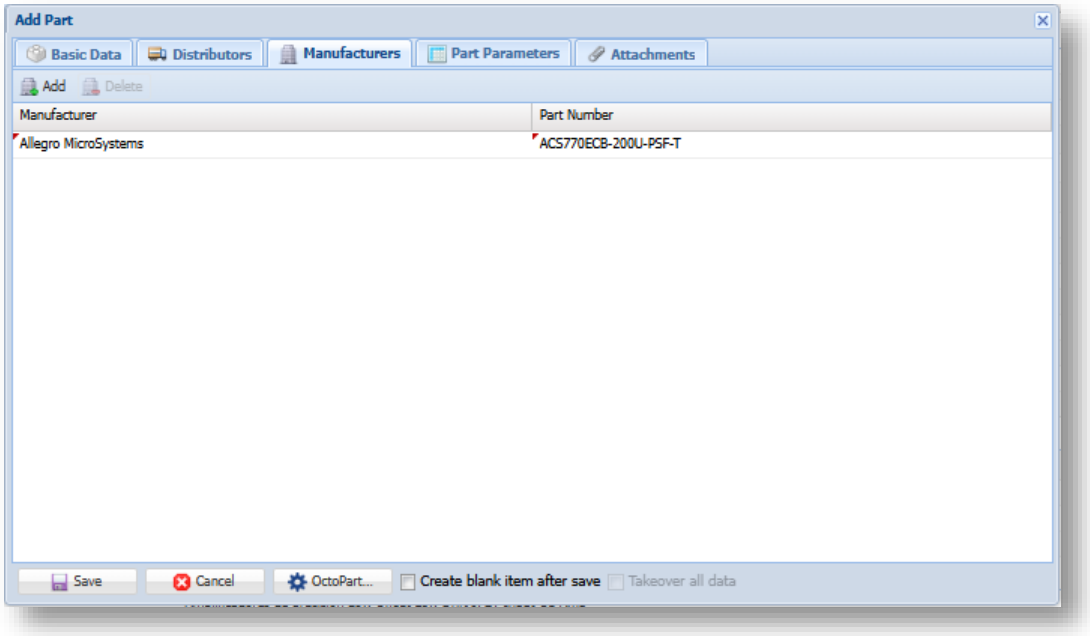



*Figure 38. UI window with manufacturers of the selected result*

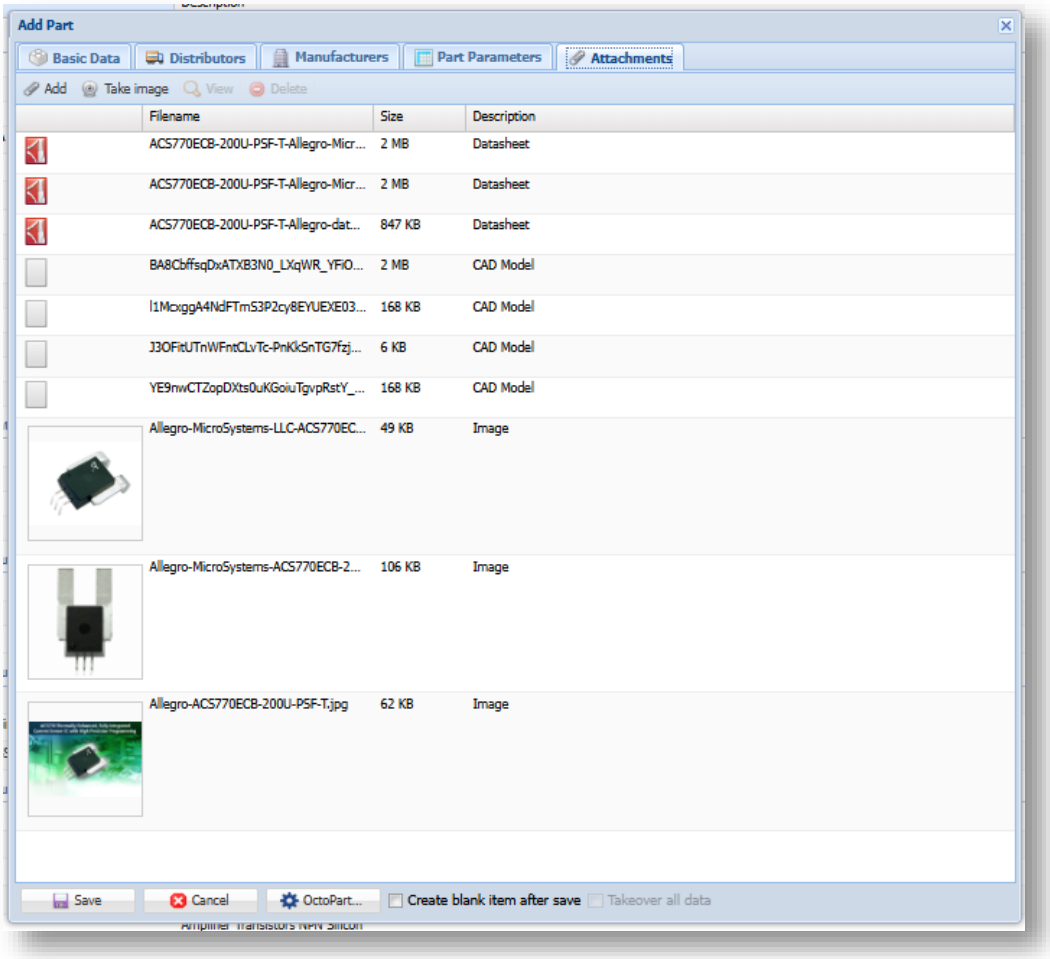



*Figure 39. UI window with attachments of the selected result*

To see the end-to-end flow more clearly:

1. The user enters the name of a component in the search bar of the Add Part window and the UI calls /api/octopart/query.
2. The backend queries Nexar GraphQL, paginates and returns a first view of the component list with image, description and price.
3. When one of the components in the list is selected, the UI requests /api/octopart/get/{mpn}.
4. The complete record arrives with all information of the selected component, and the frontend maps manufacturer, distributors and attachments.
5. Finally, the user confirms whether to save that component permanently in the PartKeepr database.

## 8.2 Configuration of Nexar Credentials (Octopart) in a Fresh PartKeepr Installation on Docker

To make all this work correctly in PartKeepr, the following steps must be followed.

First obtain credentials in Nexar by registering at https://nexar.com/api [19]

Once logged in, go to the Apps + Authorisation tab and copy the client id and client secret values as shown in the next figure.

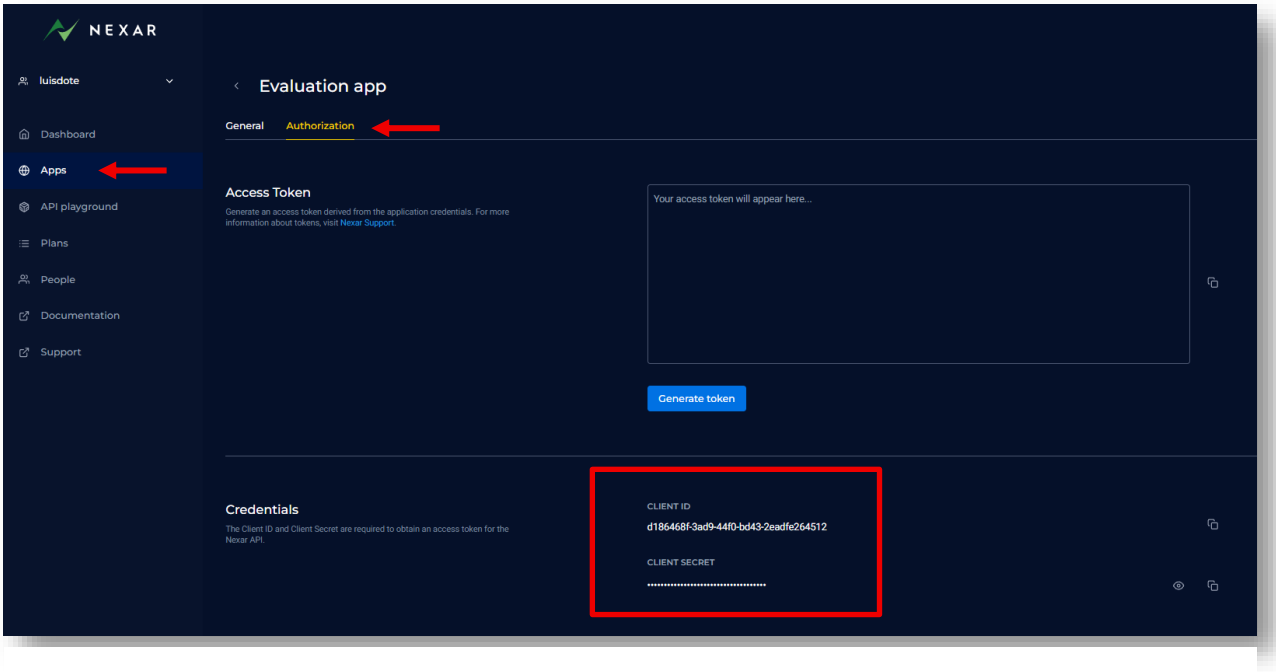



*Figure 40. Obtaining Nexar credentials*

To inject these credentials into PartKeepr, open the src folder and locate the file at src/PartKeepr/OctoPartBundle/Services/OctopartService.php.

This is the file that must be opened, and the credentials should be pasted in 'XXXX'.

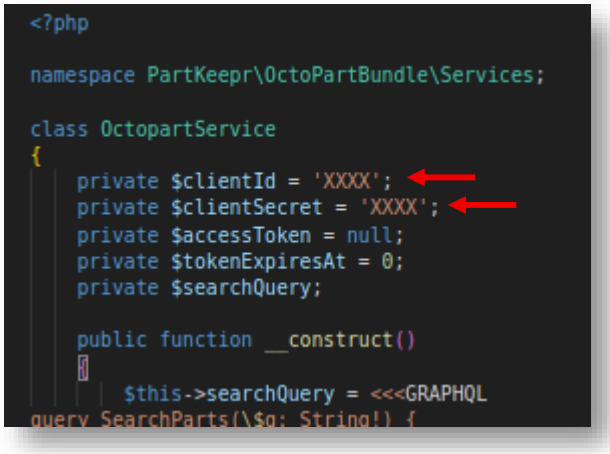



*Figure 41. Insertion of Nexar credentials*

Finally, execute the script rebuild.sh, which is included in the GitHub project.

# 9 Resolution of Uniqueness Conflicts in Storage Sub-locations

## 9.1 Context

While populating the inventory in PartKeepr, an annoying issue appeared. When creating a sub-location named “Barra 1” under Office 16, the system blocked the creation of another “Barra 1” under Office 18. From a business perspective this should be perfectly valid since names can be repeated as long as they belong to different branches of the storage tree.

## 9.2 What was discovered

The first step was to review the container logs (app/logs/dev.log) where integrity errors were visible when saving the entity.

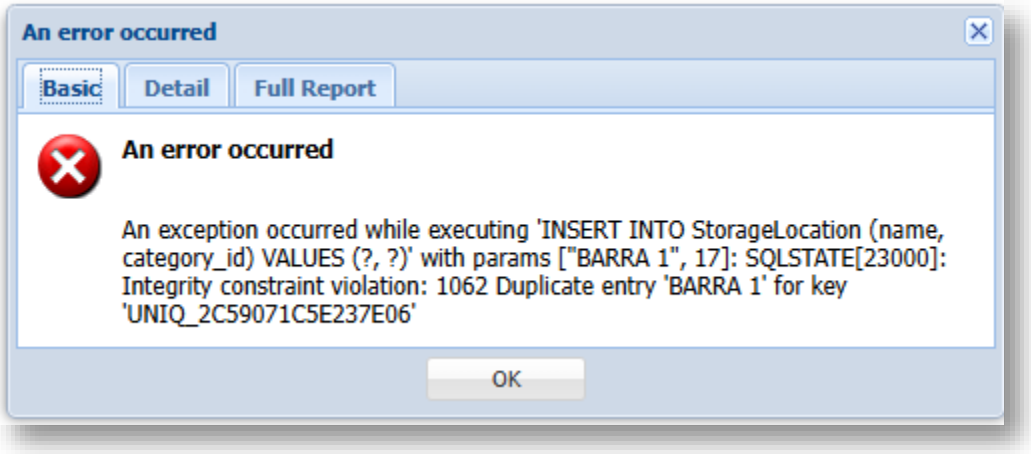



*Figure 42. Uniqueness conflict error in StorageLocation*

Next, the suspicion was confirmed in the database with SHOW CREATE TABLE StorageLocation. The table had a global unique index on name. In other words, the system treated the name as unique everywhere without considering the context (parent or category) to which it belonged.

```php
class StorageLocation extends BaseEntity
{
    /**
     * Holds the name for our storage location.
     *
     * @ORM\Column(type="string",unique=true)
     * @Groups({"default"})
     *
     * @var string
     */
    private $name;
```

*Figure 43. Previous property of name in StorageLocation*

## 9.3 Adopted Approach

The solution was to shift the uniqueness of the name field to a composite unique key that included the context. In this schema that context is expressed with category_id. The goal was to allow "Barra 1" in different offices while preventing duplicates within the same office.

The implementation followed these steps. First the uniqueness at column level (unique=true) was removed and the restriction was declared at the table level.

```php
/**
 * @ORM\Entity
 * @ORM\Table(
 *     uniqueConstraints={@ORM\UniqueConstraint(columns={"category_id", "name"})}
 * )
 * @TargetService(uri="/api/storage_locations")
 */
class StorageLocation extends BaseEntity
{
    /**
     * Holds the name for our storage location.
     *
     * @ORM\Column(type="string")
     * @Groups({"default"})
     *
     * @var string
     */
    private $name;
```

*Figure 44. Updated property in StorageLocation*

With this new restriction, Office 16 → Barra 1 and Office 18 → Barra 1 could be created without issue. On the other hand, duplicating "Barra 1" within the same office remained forbidden, which is exactly the expected behaviour.

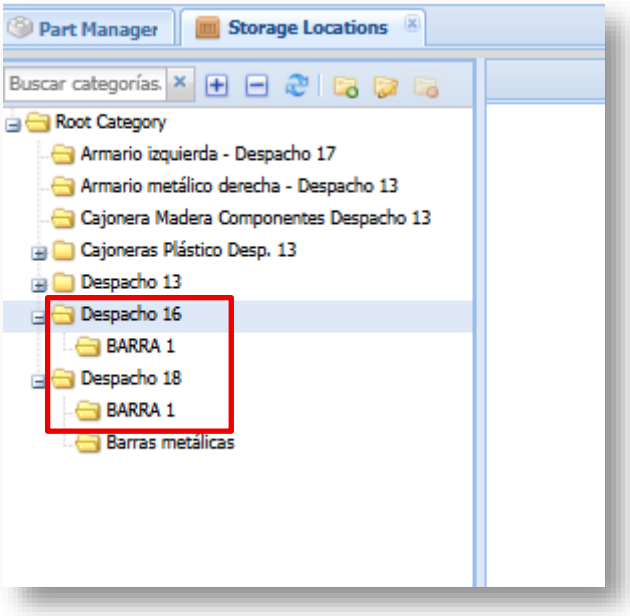



*Figure 45. UI view of duplication in StorageLocation*

# 10 Inline Editing Control in the Stock and Price Grid

## 10.1 Objective and Scope

The purpose of this improvement was to allow certain important numeric fields, such as available stock (stockLevel) and average price (averagePrice), to be modified directly from the components grid (Parts List Grid) without the need to open the full form for each item.

This aimed to address a very common situation in inventory management with large data volumes. The goal was to provide a fast and contextual way to update operational values that change frequently while ensuring that validations and permissions control who can make those modifications.

## 10.2 Initial Situation

The grid (PartsGrid) allowed editing the stockLevel cell without checking the user profile. This opened the door to accidental or unauthorised changes to these critical fields.

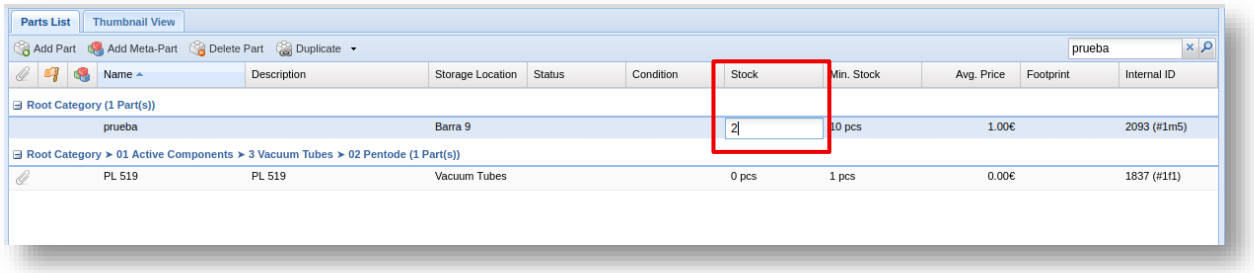



*Figure 46. Grid with editable stock field*

However, there was no possibility to modify the Avg. Price field directly from the grid.

## 10.3 Implementation

All changes were concentrated in resources/public/js/components/parts/PartsGrid.js.

In the onEdit(editor, e) handler the view retrieves the logged-in user and calculates the two flags admin and canCreateComponents.

For the stockLevel field, if the person is not an admin and does not have canCreateComponents, the grid shows a “Permission Denied” warning, restores the original value with e.record.set("stockLevel", e.originalValue), and stops the flow. Only when permission is granted and the value actually changes is the process delegated to handleStockFieldEdit(e), which keeps the familiar confirmation box (add, subtract, or set stock).

For averagePrice the pattern is the same. Without permission the system shows an alert and reverts to the previous value. With permission the change is processed. In this case the column has also been declared as a numeric editor (minimum 0 and up to four decimals) and the save operation is handled in handleAveragePriceEdit(e), which normalises the data, persists the record, and reloads the row to keep the interface consistent.

```
}, {
    header: i18n("Avg. Price"),
    dataIndex: 'averagePrice',
    align: 'right',
    renderers: [{
        rtype: "currency"
    }],  editor: {
            xtype: 'numberfield',
            minValue: 0,
            allowBlank: false,
            decimalPrecision: 4
        }
}, {
```

*Figure 47. Modified AveragePrice column*

```
onEdit: function (editor, e)
{
    const user = PartKeepr.getApplication().getUser();
    const isAdmin = user.get("admin");
    const canEdit = user.get("canCreateComponents");

    switch (e.field) {
        case "stockLevel":
            if (!isAdmin && !canEdit) {
                Ext.Msg.alert(i18n("Permission Denied"), i18n("You do not have permission to edit stock levels."));
                // Cancel the change and reset value
                e.record.set("stockLevel", e.originalValue);
                return;
            }
            if (e.value !== e.originalValue.toString()) {
                this.handleStockFieldEdit(e);
            }
            break;

        case "averagePrice":
        if (!isAdmin && !canEdit) {
            Ext.Msg.alert(i18n("Permission Denied"), i18n("You do not have permission to edit price."));
            e.record.set("averagePrice", e.originalValue);
            return;
        }
        if (e.value !== e.originalValue) {
            this.handleAveragePriceEdit(e);
        }
        break;

    default:
        break;
    }
},
```

*Figure 48. Permission control for inline editing of stock and price*

The final behaviour is clear. A user without permissions cannot alter either stock or price. They will see a denial message and the value will revert to its previous state without sending any request to the API.

An authorised user, however, can edit both fields. In stock editing the confirmation step still appears, and in price editing the save is immediate with a record reload.

## 10.4 Visual Evidence Before and After in Stock and Price Editing

The following captures illustrate the effect of this update.

The classic stock confirmation flow was executed for any account.

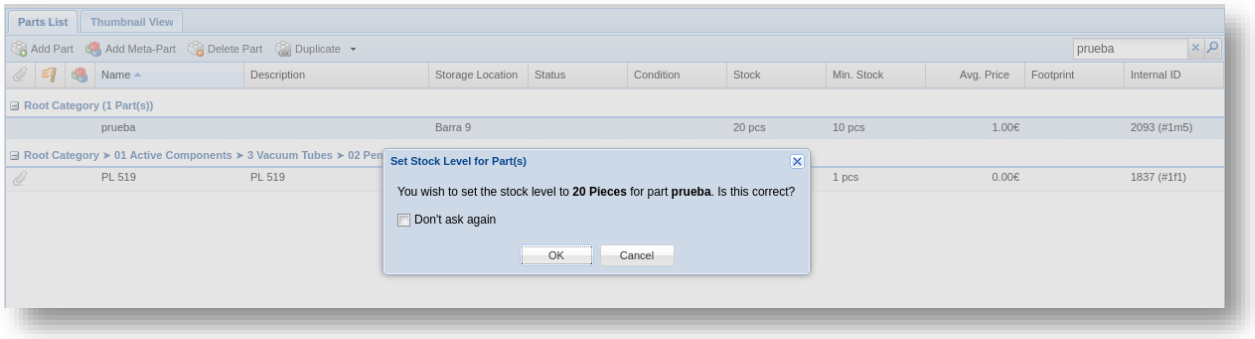



*Figure 49. Warning message when modifying stock*

With the update a numeric editor is added for Avg. Price and stock editing is preserved, but both actions are only enabled when the user is authorised.

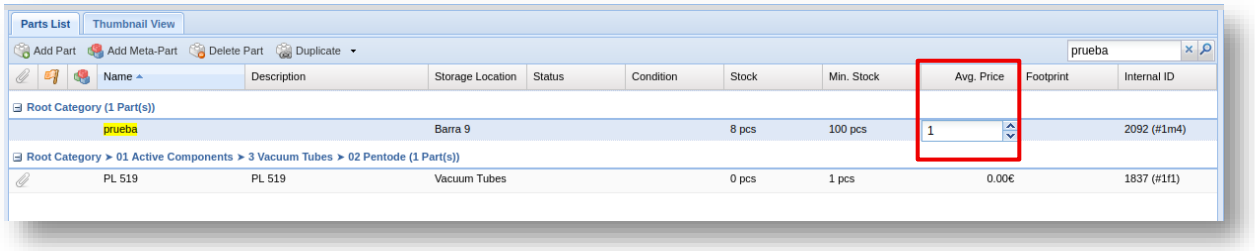



*Figure 50. Updated grid*

The following captures show the warning messages displayed to unauthorised users attempting to modify the stock and average price fields.

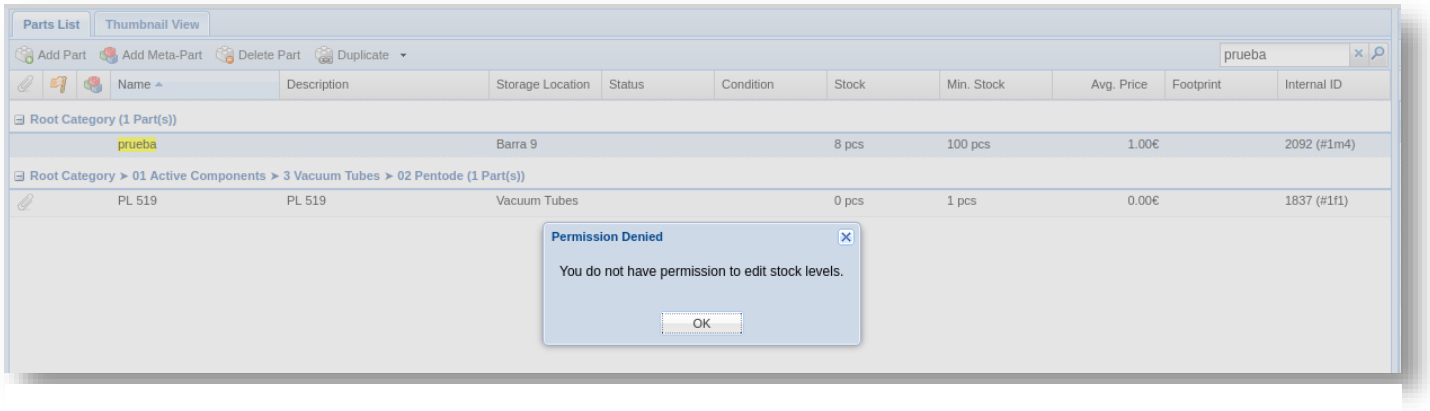



*Figure 51. Message for unauthorised user attempting to modify stock*

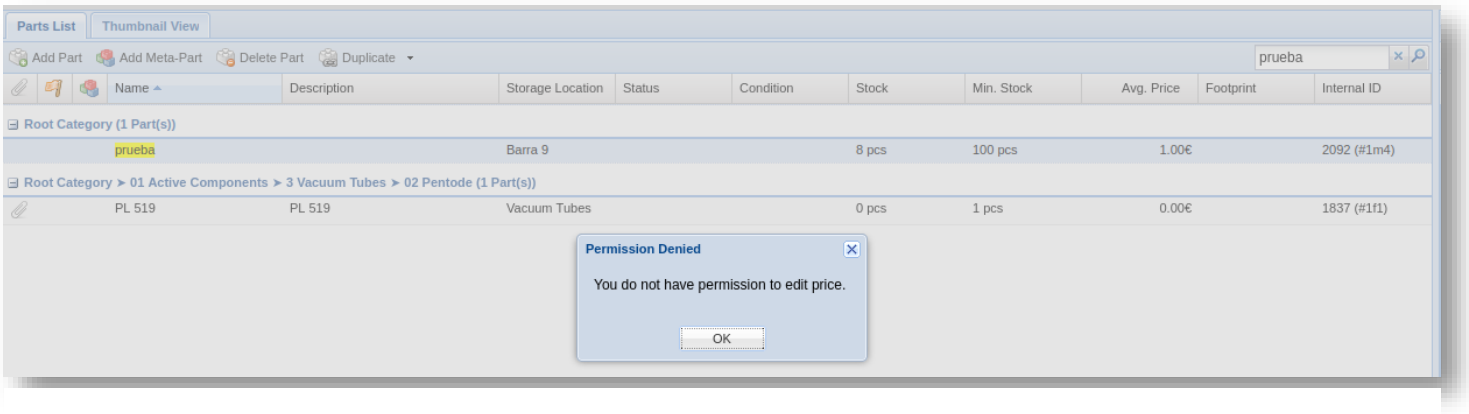



*Figure 52. Message for unauthorised user attempting to modify price*

# 11 Search Improvement Non-destructive Highlighting in the Grid

## 11.1 Objective

The goal was to make the Parts List search tool truly useful in daily work without disorienting the user or altering the inventory view. The idea was that when typing, rows should not disappear and groups should not collapse. Instead, all occurrences of the term in the visible columns (name, description, location and so on) are highlighted in yellow. This allows the user to quickly scan where the matches appear while counters, groups and pagination remain intact.

## 11.2 Initial situation

The original search acted by filtering the Store or forcing a data reload. This had undesirable side effects. When working with grouping or pagination, some rows disappeared or moved to another page, and the user lost track of what they were viewing. In addition, even if the term existed within a row, the interface did not indicate where the match was located (name, description and so on), which meant opening each record or reading it fully.

## 11.3 Term capture

During the initialisation of the grid (initComponent()), the searchField is hooked. Each time the user types or deletes, a listener triggers a repaint of the grid with this.getView().refresh(). No API calls are made and the Store is not modified. During repaint, the renderers read the current search value and apply the highlighting helper to each visible cell.

The behaviour is immediate. If the field is cleared, the next refresh removes the highlight and everything returns to its original state. If needed, a small debounce could be added to smooth refreshes.

```
this.searchField.on('change', () => {
    this.getView().refresh();
});
```

*Figure 53. Search listener that repaints the grid to apply highlighting*

## 11.4 Highlight function

The helper highlightSearch(value, searchTerm) paints the matches of the search term inside each cell. It does not touch the Store data or call the API. It only returns decorated HTML so that the grid displays those matches with a yellow background.

Step by step:

1. **Case with no term**. If searchTerm is empty or null, the function returns the original content protected with Ext.String.htmlEncode(value) to prevent characters like < or & from breaking the markup.
2. **Safe preparation of the term**. If a term exists, it is first escaped with Ext.String.escapeRegex(searchTerm) to neutralise any special characters such as dots, brackets or parentheses. This prevents errors and ensures the search remains accurate.
3. **Case-insensitive search**. A RegExp is created with flags gi so that all matches are highlighted regardless of case.
4. **Protection and replacement**. Before replacing, the original cell text is re-encoded with htmlEncode. On this safe HTML, replace(regex, '$1') is applied, wrapping each match in a span with yellow background.
5. **Result**. The function returns an HTML string ready for the grid to render. If the user clears the field, the next refresh restores the cell without highlighting.

```
/**
 * Highlights search term inside grid cells.
 */
highlightSearch: function (value, searchTerm) {
    if (!searchTerm) {
        return Ext.String.htmlEncode(value);
    }
    var escapedSearch = Ext.String.escapeRegex(searchTerm);
    var regex = new RegExp('(' + escapedSearch + ')', 'gi');
    return Ext.String.htmlEncode(value).replace(regex, '<span style="background-color: yellow;">$1</span>');
},
```

*Figure 54. Function that safely highlights matches inside cells*

## 11.5 Column Integration

To display the search term inside cells, the change was made in the renderers of each textual column of the PartsGrid.

The idea is simple. Each column formats its value as usual, and then the highlightSearch() helper wraps the matches with the highlight span.
To achieve this, resources/public/js/components/parts/PartsGrid.js was modified. Inside defineColumns(), function renderers were added to columns such as Name and Description and can be extended to Storage Location, Status, Condition, Footprint and others.)

```
}, {
    header: i18n("Name"),
    dataIndex: 'name',
    flex: 1,
    minWidth: 150,
    renderer: (value) => {
        const searchTerm = this.searchField ? this.searchField.getValue() : "";
        return this.highlightSearch(value, searchTerm);
    }
}, {
    header: i18n("Description"),
    dataIndex: 'description',
    flex: 2,
    minWidth: 150,
    renderer: (value) => {
        const searchTerm = this.searchField ? this.searchField.getValue() : "";
        return this.highlightSearch(value, searchTerm);
    }
},
```

*Figure 55. Highlight applied in text column renderers (Name and Description)*

## 11.6 Why This Solution Is Suitable

The highlight search is user friendly because it does not break the context. Nothing is hidden or rearranged. Groups, counters and pagination remain exactly as they are. Only the matching text is painted in yellow so the user's eye is drawn directly to it. When the term is deleted, the grid repaints and everything returns to normal. No hidden filters remain.

It is also a very lightweight improvement. Everything happens in the browser. The search listener triggers a grid refresh that applies the highlight in visible cells. There are no reloads, no extra queries and no heavy background processing. If the user types quickly,

the only cost is rendering.

Another advantage is that it is completely isolated from the backend. Doctrine, endpoints and the ExtJS Store are untouched. The data remains unchanged and the API receives no extra traffic. This simplifies maintenance with fewer points of failure and makes it easier to adjust the experience. For example, changing the highlight colour or adding or removing columns requires only a CSS rule or a single line in the renderer.

In summary, the search is now clearer and more usable without cost in complexity, performance or stability.

## 11.7 Test performed

The following test shows the Parts List after typing transistor in the search field.

The grid does not filter or hide rows. It keeps categories and groups open and highlights in yellow all matches in visible columns such as Name and Description. The term is visible in several cells across groups without altering counters or pagination.

In the bottom bar, the classic PartKeepr search expression can be seen (“name like… OR description like… OR comment like… OR internalPartNumber like…”). However, the behaviour of this improvement is non-destructive. It only changes the cell HTML to improve readability. The Store and API queries remain unchanged.

This test validated three aspects. Matches are case insensitive. Highlighting is applied to all appearances inside each cell. When the term is deleted the grid returns to its original state on the next refresh without reloads or loss of context.

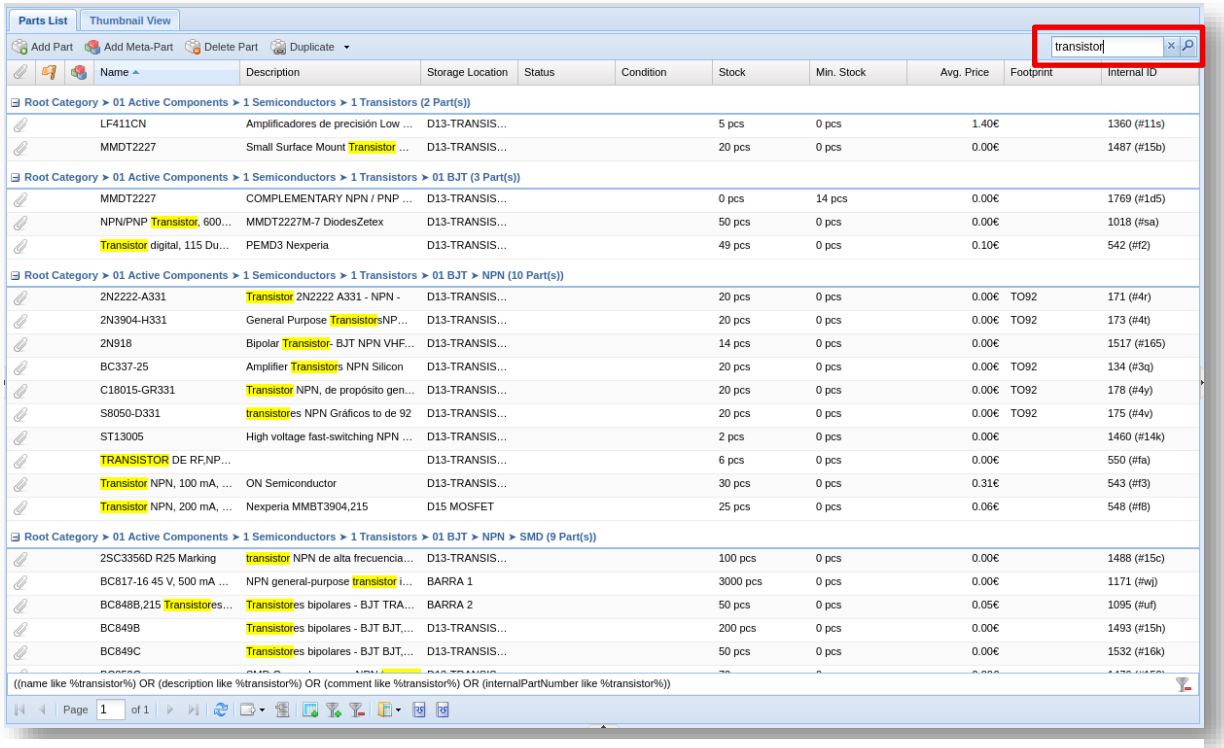



*Figure 56. Highlighting of matches in the search field*

## 11.8 Result

The search tool now guides the user's eye. Everything remains visible and the exact matches are marked in each row. The change is minimal, local and maintainable, involving only a couple of methods and a CSS style. Usability improves without touching the data logic.

# 12 Category search in the tree

## 12.1 Objective and scope

In installations with hundreds or thousands of categories, locating a specific folder within the tree was slow. Nodes had to be opened manually until the correct one was found. The aim of this improvement was to add a search bar in the categories tab to find any folder by name or by its path (categoryPath) while keeping both the hierarchy and the natural order of the tree. The change is compatible with the usual behaviour of the editor and does not alter the data model.

## 12.2 Initial situation

The Categories window in PartKeepr displayed the full tree but did not offer any search field. In deep trees, locating a branch required repetitive expanding and collapsing of nodes, which consumed time and increased the chance of errors.

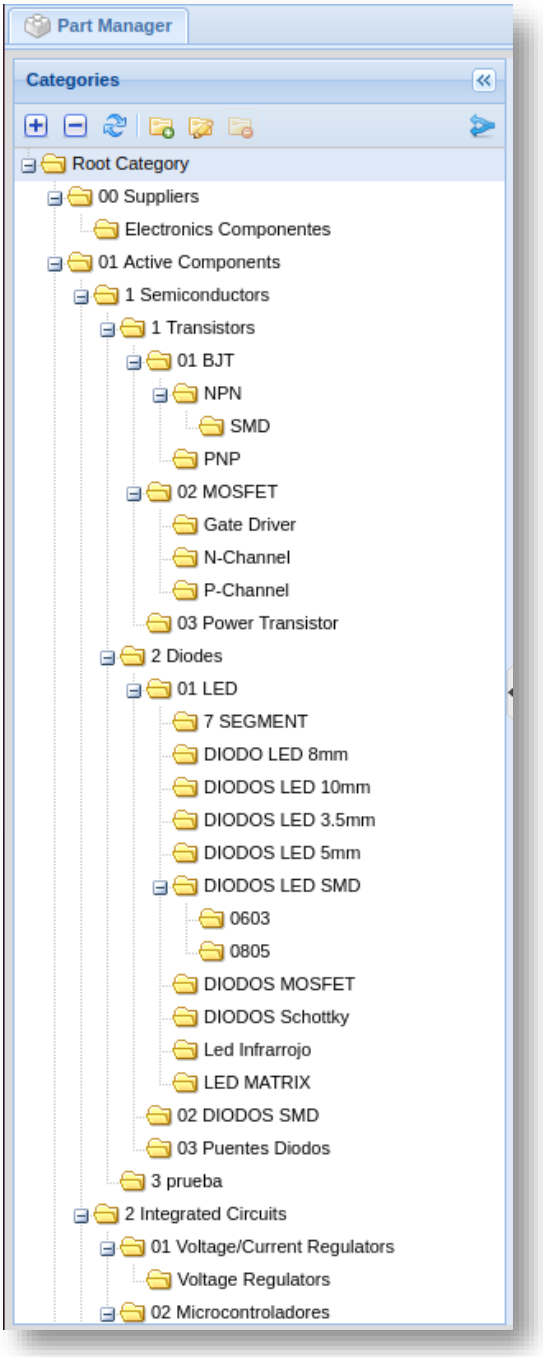



*Figure 57. Categories window without search field (original)*

## 12.3 Adopted Design and Operation

To enable searching within the category tree without touching the data model, I added a specific REST action in the CategoryBundle.

The idea was simple: expose an endpoint that accepts a search term and returns only those categories whose name or full path (categoryPath) contain that text, maintaining the natural reading order of the tree.

Specifically, the SearchCategoriesAction.php class was created. This action receives the q parameter by query string, builds a query with Doctrine QueryBuilder, and filters with a partial condition (LIKE %q%) on name and categoryPath. The result is sorted by categoryPath in ascending order, which preserves the hierarchy seen by the user (matching the existing nested set logic). If q arrives empty, the action returns the entire set, also sorted.

The action was registered as a service in the actions.xml configuration file, under the identifier partkeepr.category.search, injecting doctrine as a dependency. The public route was defined in the same XML:

- Method and route: GET /api/part_categories/search
- Parameters: q (free text)
- Output: JSON collection with matching categories, sorted by categoryPath

This approach has several advantages for a TFG: it is non-intrusive (it does not modify entities or repositories), the query is parameterised (no SQL concatenation, avoiding injections), and it is encapsulated as an additional service of the bundle, alongside the existing actions (get root, move nodes, etc.).

```xml
<?xml version="1.0" ?>

<container xmlns="http://symfony.com/schema/dic/services"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://symfony.com/schema/dic/services
                               http://symfony.com/schema/dic/services/services-1.0.xsd">

    <services>
        <!-- Action to obtain the root node -->
        <service id="partkeepr.category.get_root_node"
                 class="PartKeepr\CategoryBundle\Action\GetRootNodeAction">
            <argument type="service" id="doctrine"/>
        </service>

        <!-- Action to move categories -->
        <service id="partkeepr.category.move"
                 class="PartKeepr\CategoryBundle\Action\MoveAction">
            <argument type="service" id="api.data_provider"/>
            <argument type="service" id="api.iri_converter"/>
            <argument type="service" id="doctrine"/>
        </service>

        <!-- Action to search categories -->
        <service id="partkeepr.category.search"
                 class="PartKeepr\CategoryBundle\Action\SearchCategoriesAction">
            <argument type="service" id="doctrine"/>
        </service>

         <!-- Subscriber to reorder newly created node to first child -->
        <service id="partkeepr.category.category_tree_subscriber"
                 class="PartKeepr\CategoryBundle\EventListener\CategoryTreeSubscriber">
            <tag name="doctrine.event_subscriber"/>
        </service>
    </services>
</container>
```

*Figure 58. Registration of the service and REST endpoint for category search in actions.xml*

```php
/**
 * Returns a filtered list of categories by name or path.
 */
class SearchCategoriesAction
{
    use ActionUtilTrait;

    /**
     * @var ManagerRegistry
     */
    private $manager;

    public function __construct(ManagerRegistry $manager)
    {
        $this->manager = $manager;
    }

    /**
     * Retrieves a collection of PartCategory matching the query.
     *
     * @param Request $request
     *
     * @throws RuntimeException
     *
     * @return array|\Traversable
     */
    public function __invoke(Request $request)
    {
        // We extract the resource type (PartCategory)
        list($resourceType) = $this->extractAttributes($request);

        // We retrieve the Doctrine repository
        $repository = $this->manager->getRepository($resourceType->getEntityClass());

        // We read the query parameter "q"
        $term = trim($request->query->get('q', ''));

        // We build the QueryBuilder
        $qb = $repository->createQueryBuilder('c')
            ->orderBy('c.categoryPath', 'ASC');

        if ($term !== '') {
            $qb->andWhere('c.categoryPath LIKE :term OR c.name LIKE :term')
               ->setParameter('term', '%' . $term . '%');
        }

        // We execute and return the result
        return $qb->getQuery()->getResult();
    }
}
```

*Figure 59. Implementation of the category search endpoint (SearchCategoriesAction.php)*

**Frontend integration in the category tree (ExtJS)**

To avoid endless tasks when searching for a folder in the tree, a search field was added in the toolbar of the CategoryEditorTree. The change is purely presentational. It does not alter the structure of the tree or its usual events (selection, drag and drop) and coexists with the other controls without taking up space.

The tree now has an explicitly defined TreeStore. This way, when activated, we point to the search endpoint and deactivate any client-side sorting (folderSort: false, sorters: []). This ensures that the order displayed is exactly the same as the one coming from the server (or the nested set), avoiding visual jumps while typing.

For the search field in the toolbar, in createToolbar() I incorporated an Ext.form.field.Trigger with the placeholder 'Search categories...'. This component is convenient for two reasons: it allows typing and also has a 'clear' icon that deletes the text with one click. When clicked, the value is cleared and the entire tree is restored. The field listens for keyboard events (enableKeyEvents: true) and applies a 300 ms debounce to avoid triggering unnecessary loads while the user is typing.

```
createToolbar: function () {
    var me = this;
    this.toolbarSearchField = Ext.create('Ext.form.field.Trigger', {
        emptyText: i18n('Search categoriess...'),
        width: 120,
        triggerCls: 'x-form-clear-trigger',
        enableKeyEvents: true,
        onTriggerClick: function () {
            this.setValue('');
            me.clearFilter();
        },
        listeners: {
            change: function (field, value) {
                me.filterTreeByText(value);
            },
            buffer: 300
        }
    });
```

*Figure 60. Implementation of the search field in the category tree toolbar*

The toolbar is created inside initComponent(), callParent() is invoked, and the existing tree handlers for selection and drag and drop are preserved. The result is a lightweight search bar, localised with i18n, 120 px wide so as not to displace the Add, Edit and Delete buttons, and able to find a category in seconds.

## 12.4 Result

The category tree now includes a discreet and efficient search tool. Finding a specific folder requires only typing a few letters. The hierarchy is preserved, the response is fast, and users avoid navigating node by node. In terms of usability, the improvement is significant with low technical cost and no impact on the data domain.

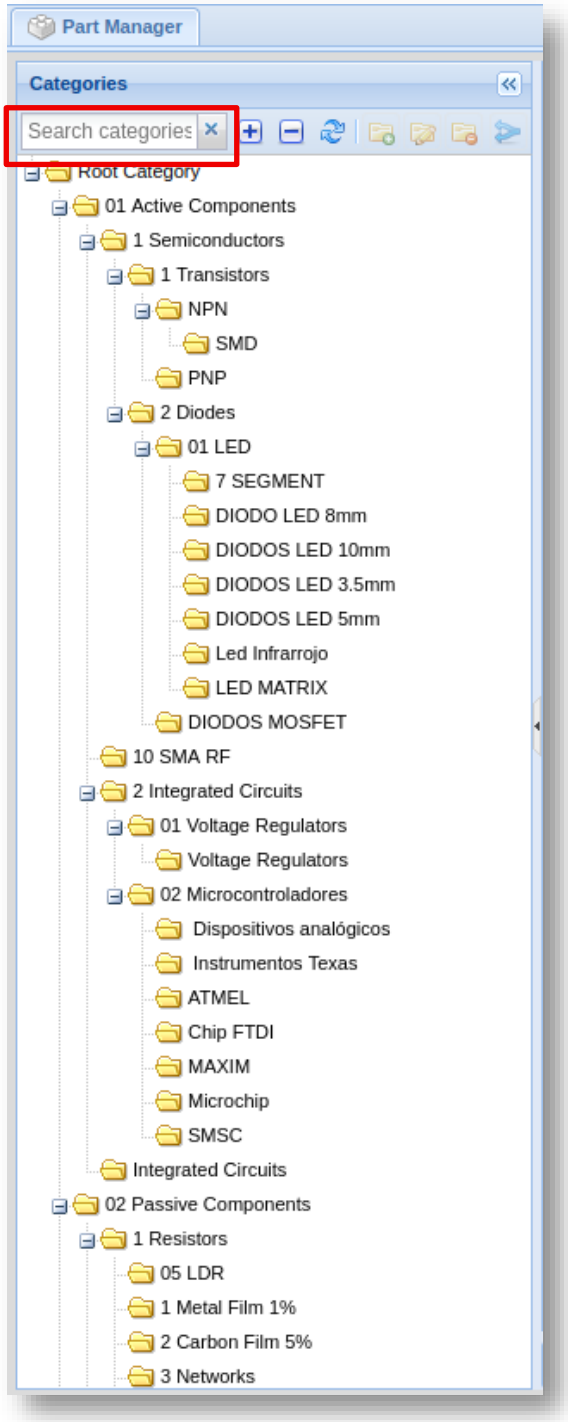



*Figure 61. Final result of the category search engine*

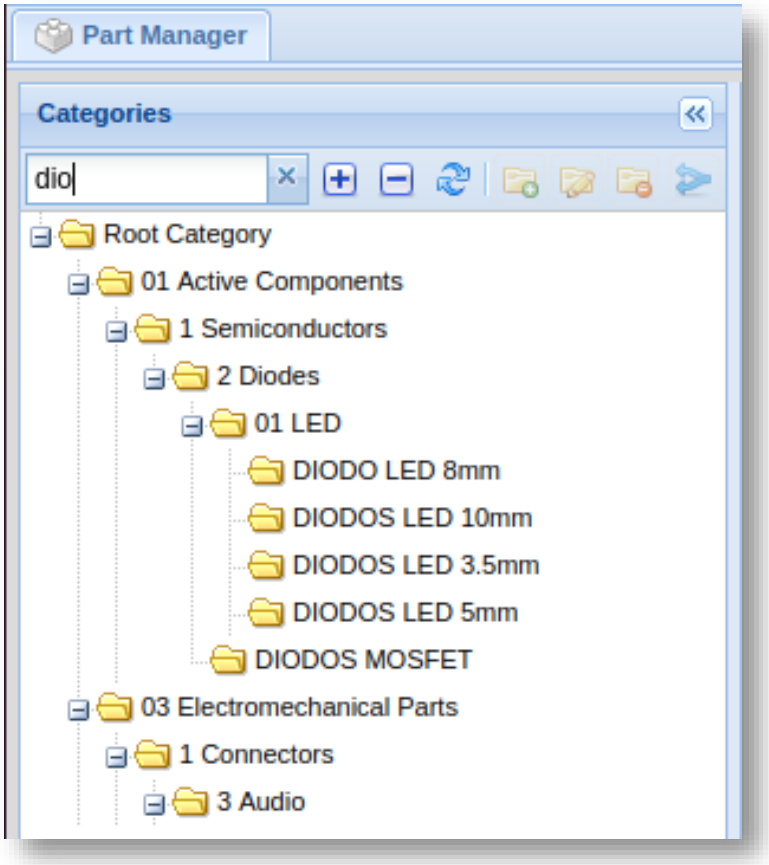



*Figure 62. Search test in the category search engine*

# 13 About this installation Dialogue with link to the upgrade kit on Github

To replace the old Patreon Status notice, which no longer provided useful information, an information box about the installation was added. It is accessible from the top bar with the text About this installation. When opened, it displays a short message confirming that the instance has been updated and extended as part of this project, and offers a direct button to the GitHub repository (upgrade-kit folder) where the scripts and files needed to apply or review the update are published.

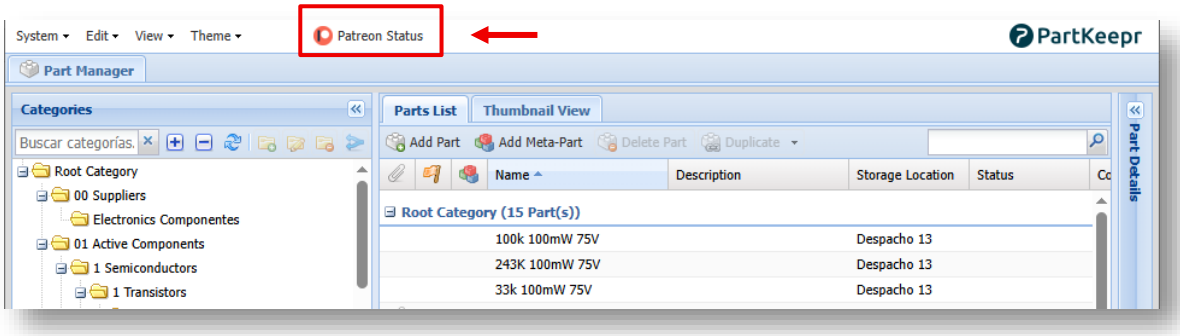



*Figure 63. Old Patreon Status notice*

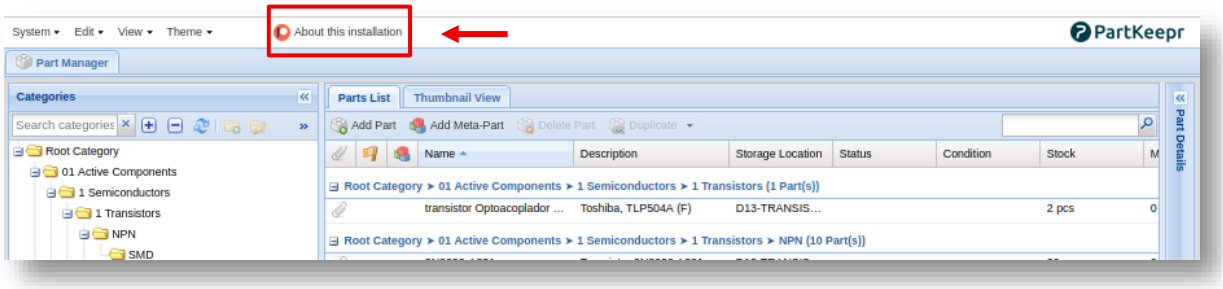



*Figure 64. New About this installation notice*

The implementation was done with a lightweight ExtJS component, keeping internationalisation through i18n.

PartKeepr.Components.PatreonStatusDialog was redefined as a window (Ext.window.Window) with the title About this installation, descriptive HTML content, and two footer actions:

- **View on GitHub opens the upgrade-kit in a new tab.**
- **Close**.

This change improves traceability and support. Any user can immediately see that the installation runs on the maintained fork and has a clear entry point to the technical material such as scripts, docker-compose and deployment artefacts without needing to navigate through the project tree.

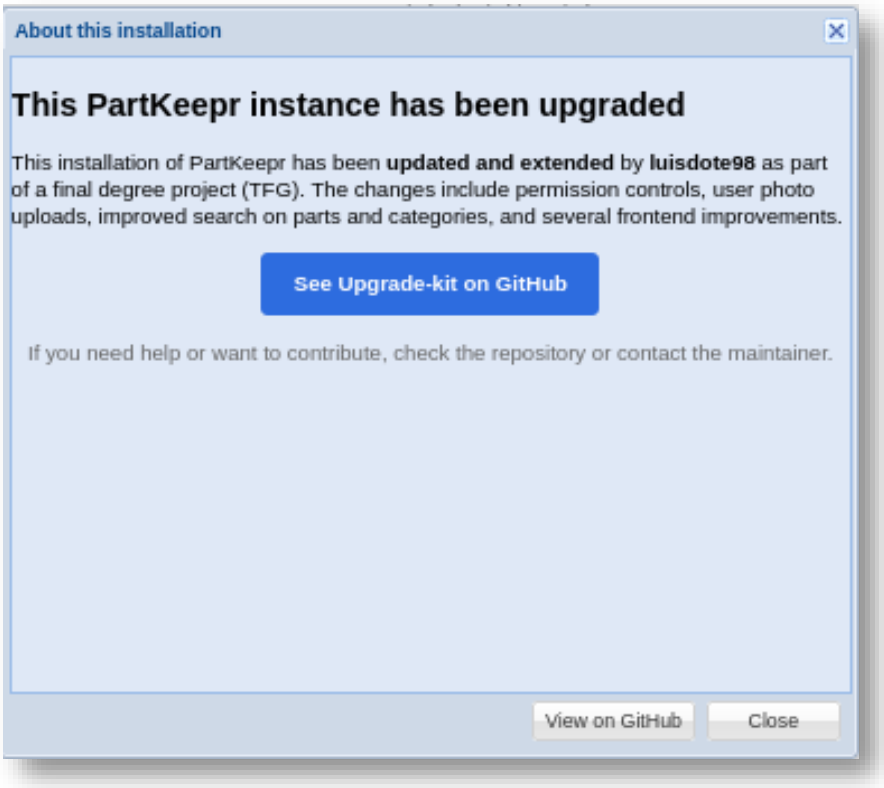



*Figure 65. New window about this installation*

# 14 Deployment and dockerisation of PartKeepr

## 14.1 Objective and scope

This section describes a reproducible deployment of PartKeepr with Docker Compose, both for fresh installations and for restoring from a database backup. The text is based on an operational guide verified step by step. [20]

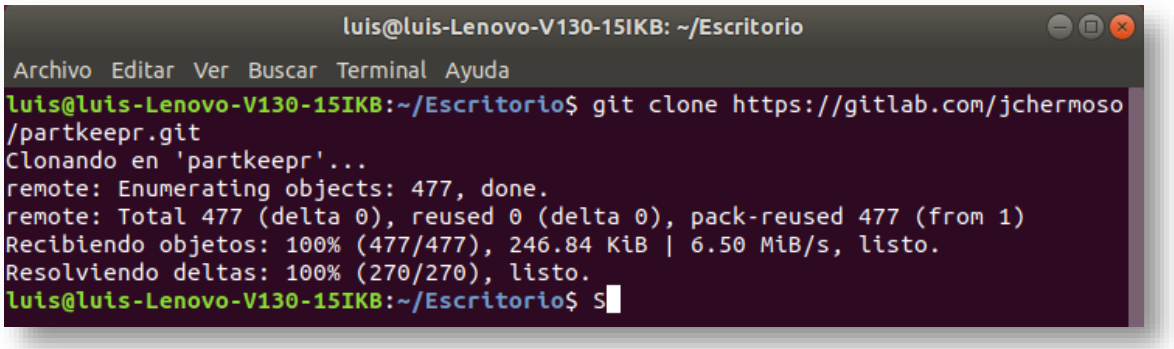
## 14.2 Clean installation from scratch

For a clean installation from zero, the PartKeepr repository must be cloned and the following steps carried out:

- To begin with, git must be installed since it is required to clone repositories:
  - sudo apt update
  - sudo apt install git -y

- Next, Docker must be installed:
  - sudo apt install docker-compose -y
  - sudo usermod -aG docker $USER
  - newgrp docker

- The following step is to move to the Desktop, open the Terminal, and download the PartKeepr source code from GitLab:
  - git clone https://gitlab.com/jchermoso/partkeepr.git




- Once the repository is downloaded and extracted, the necessary permissions must be set on the partkeepr folder.
  - chmod -Rf 777 partkeepr/

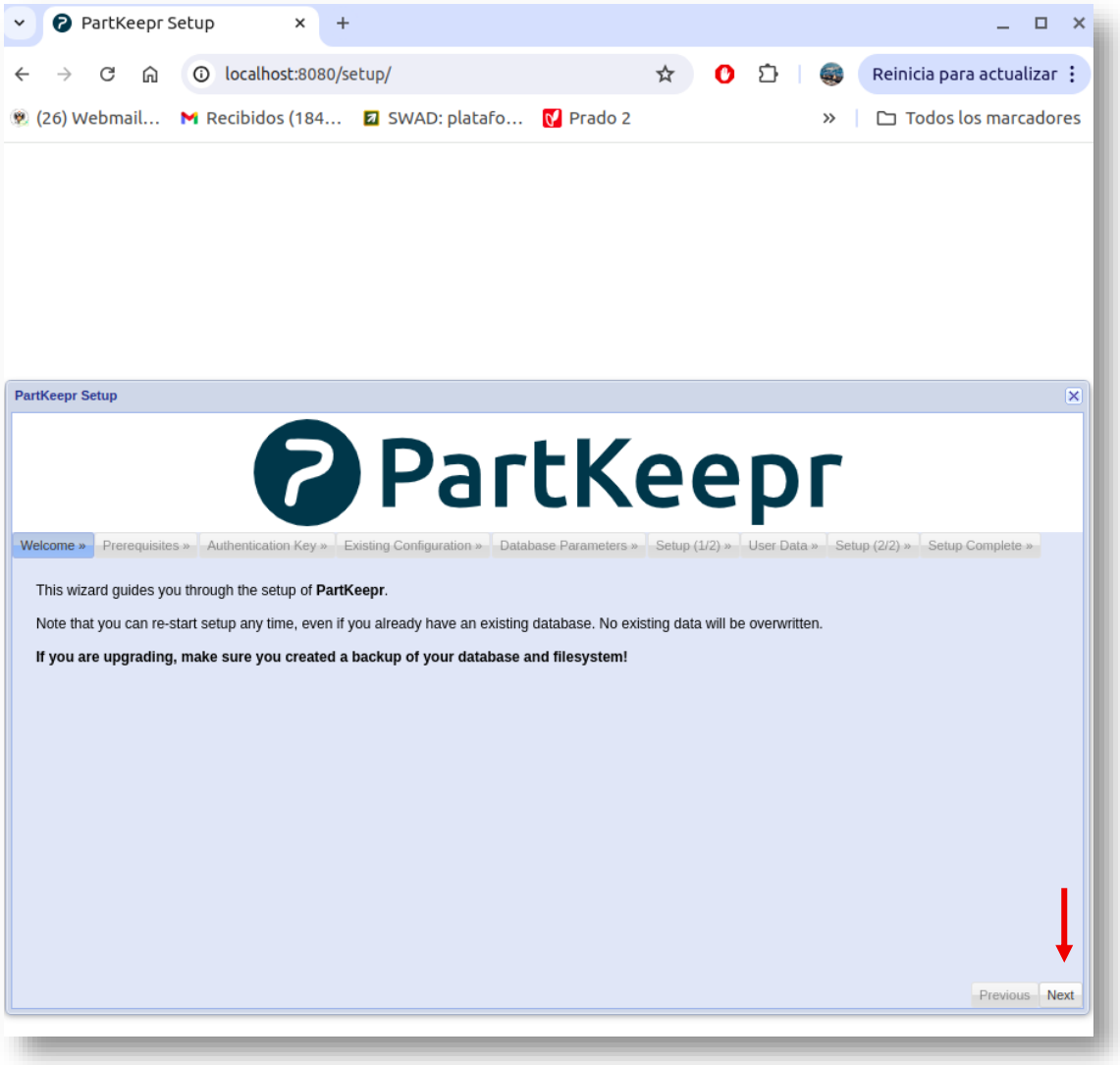
- After adjusting the permissions, enter the partkeepr folder and update the repository with the latest changes.
    - cd partkeepr
    - git pull

```
luis@luis-Lenovo-V130-15IKB:~/Escritorio$ cd partkeepr/
luis@luis-Lenovo-V130-15IKB:~/Escritorio/partkeepr$ git pull
Ya está actualizado.
luis@luis-Lenovo-V130-15IKB:~/Escritorio/partkeepr$
```

- At this stage, the PartKeepr Docker container can be launched. Always ensure you are inside the partkeepr folder.
    - docker-compose up -d

```
luis@luis-Lenovo-V130-15IKB:~/Escritorio/partkeepr$ docker-compose up -d
Starting partkeepr_database_1 ... done
Starting partkeepr_partkeepr_1 ... done
Starting partkeepr_cronjob_1   ... done
luis@luis-Lenovo-V130-15IKB:~/Escritorio/partkeepr$
```

- From here, PartKeepr can be configured by opening the browser at http://localhost:8080/setup

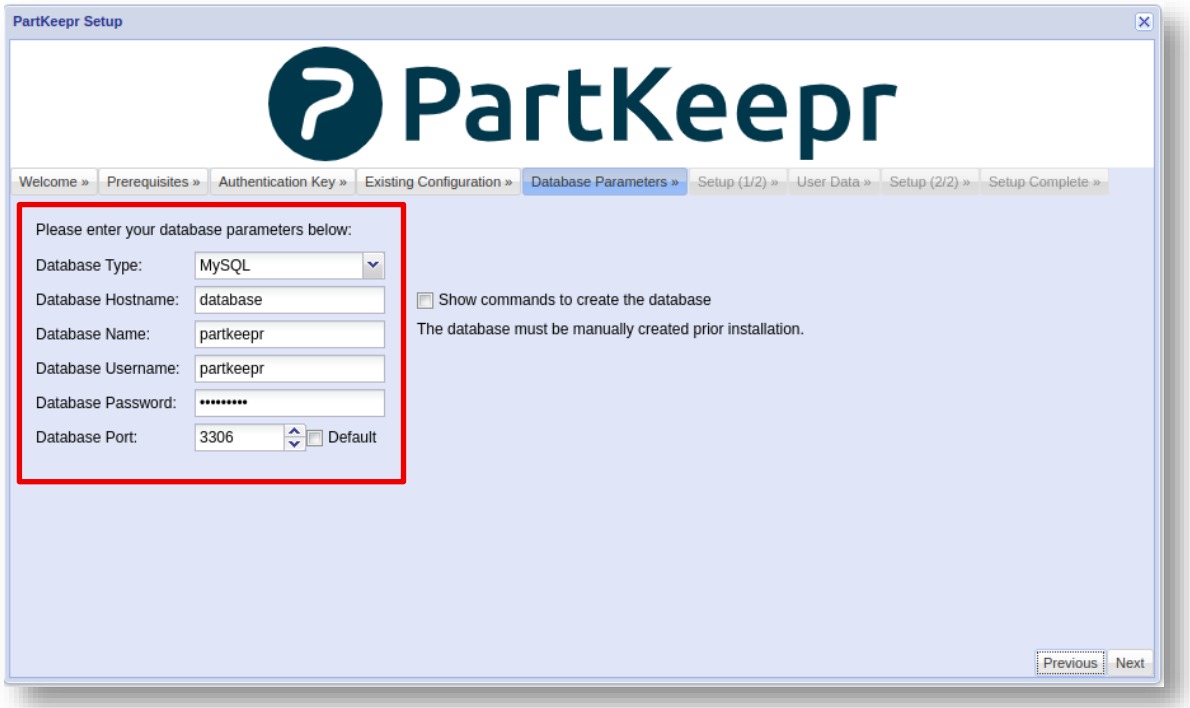



- When pressing the Next button in the bottom right corner, the Authentication Key page will appear. This key can be obtained by executing the following command:
    - docker-compose exec partkeepr cat app/authkey.php

```
luis@luis-Lenovo-V130-15IKB:~/TFG/partkeepr$ docker-compose exec partkeepr cat a
pp/authkey.php
<?php
/**
 * Your auth key is: DNGPIGHHIJLBOLALGLHPKMJOKMCKKHQE
 *
 * Copy and paste the auth key in order to proceed with setup
 */
```

- Copy the key and paste it into the Setup window.


- Proceeding further with Next leads to the Database Parameters page, where predefined values are already show.



- Advancing again brings up the User Data page, where username, password and email must be entered.


- Finally, the Setup Complete window is reached. After pressing Submit, PartKeepr is installed and can be accessed via http://localhost:8080

## 14.3 Restoration of an existing instance

When an SQL dump (for example backup_part.sql) and a data/ folder with attachments and images already exist, the entire state can be recovered in the new Dockerised installation.

- First of all, the SQL dump must be copied to the database container. Start by locating the real name of the database container with the command:
  - docker ps

```
CONTAINER ID   IMAGE                    NAMES
44b277b280d7   mhubig/partkeepr:latest  partkeepr_cronjob_1
a86f38a46ac9   mhubig/partkeepr:latest  partkeepr_partkeepr_1
634b20737e61   mariadb:10.0             partkeepr_database_1
```

In this case the container is called partkeepr_database_1 and comes from the mariadb image

- Once the container is identified, copy the backup file into it:
    - docker cp ~/ruta/backup_part.sql partkeepr_database_1:/backup_part.sql

```
luis@luis-Lenovo-V130-15IKB:~/Escritorio/partkeepr$ docker cp ../backup_part.sql
 partkeepr_database_1:/backup_part.sql
Successfully copied 547kB to partkeepr_database_1:/backup_part.sql
luis@luis-Lenovo-V130-15IKB:~/Escritorio/partkeepr$
```

- After that, access the database container:
    - docker exec -it partkeepr_database_1 bash

- At this step, use the credentials specified in docker-compose.yml (user partkeepr, password partkeepr) to restore the database:
    - mysql -u partkeepr -p partkeepr < /backup_part.sql

```
luis@luis-Lenovo-V130-15IKB:~/TFG/partkeepr$ docker exec -it partkeepr_database_
1 bash
root@39ea708a676e:/# mysql -u partkeepr -p partkeepr < /backup_part.sql
Enter password:
root@39ea708a676e:/# exit
```

- The next task is to copy the data folder inside the PartKeepr container. Identify the container, which in this case is partkeepr_partkeepr_1:

```
tkeepr$ docker ps
 CREATED       STATUS         PORTS                                    NAMES
 4 days ago    Up 17 hours    80/tcp                                   partkeepr_cronjob_1
 4 days ago    Up 17 hours    0.0.0.0:8080->80/tcp, :::8080->80/tcp    partkeepr_partkeepr_1
 4 days ago    Up 17 hours    3306/tcp                                 partkeepr_database_1
```

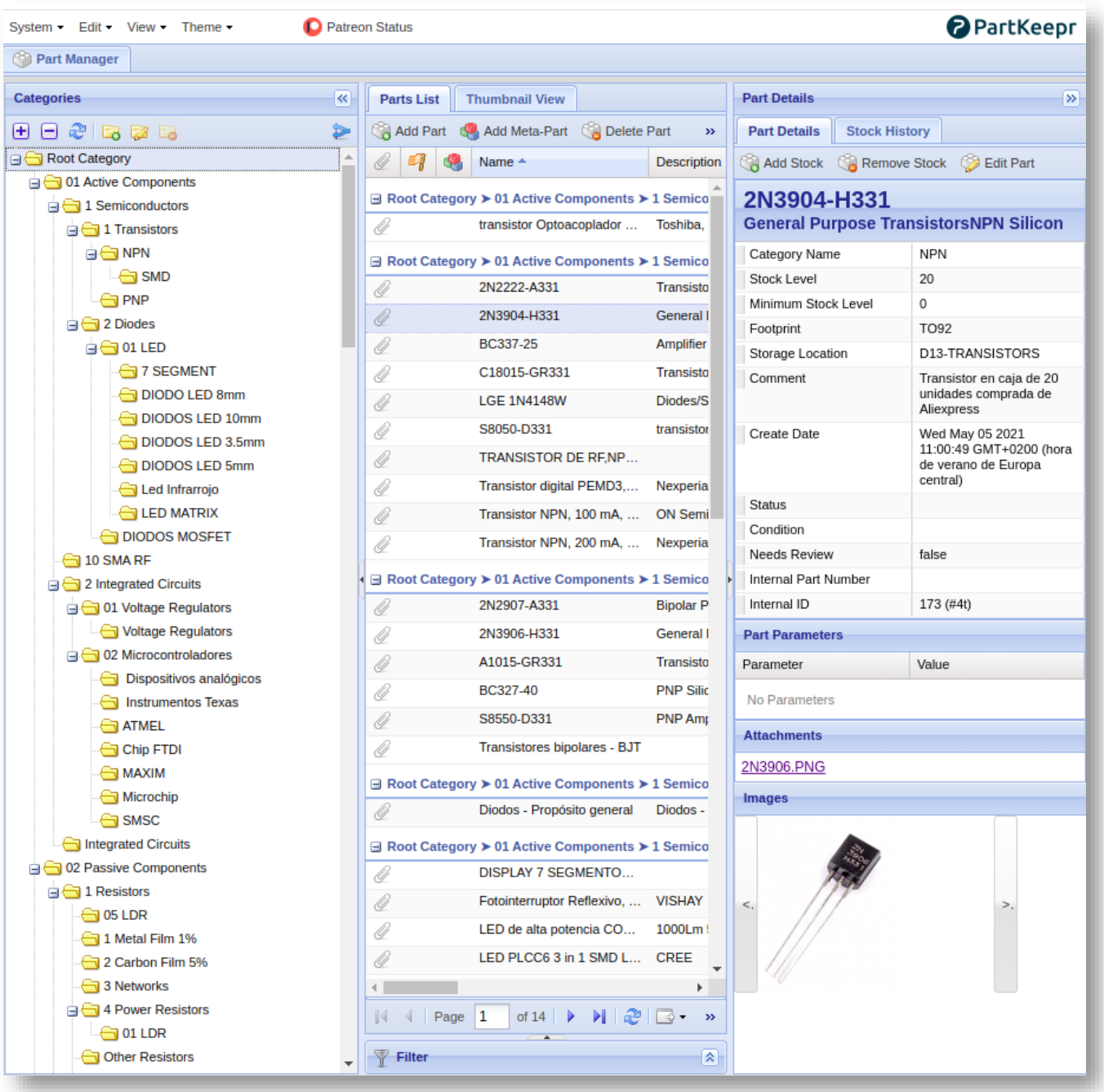
- Copy the data folder with the following command. It is important to include /. to ensure the entire contents are transferred and old files are replaceds:
    - docker cp ~/ruta/data/. partkeepr_partkeepr_1:/var/www/html/data

```
luis@luis-Lenovo-V130-15IKB:~/TFG/partkeepr$ docker cp ~/Escritorio/data/. partk
eepr_partkeepr_1:/var/www/html/data
Successfully copied 2.01GB to partkeepr_partkeepr_1:/var/www/html/data
```

- Once these actions are completed, the previous database is restored in the new installation and the application runs with the old data in the local environment. To finalise, execute the following commands.
    - docker-compose down
    - docker-compose up -d

```
luis@luis-Lenovo-V130-15IKB:~/TFG/partkeepr$ docker-compose down
Stopping partkeepr_cronjob_1   ... done
Stopping partkeepr_partkeepr_1 ... done
Stopping partkeepr_database_1  ... done
Removing partkeepr_cronjob_1   ... done
Removing partkeepr_partkeepr_1 ... done
Removing partkeepr_database_1  ... done
Removing network partkeepr_default
luis@luis-Lenovo-V130-15IKB:~/TFG/partkeepr$ docker-compose up -d
Creating network "partkeepr_default" with the default driver
Creating partkeepr_database_1 ... done
Creating partkeepr_partkeepr_1 ... done
Creating partkeepr_cronjob_1   ... done
```

- At this point, PartKeepr can be accessed through the browser and the restored data will be available.



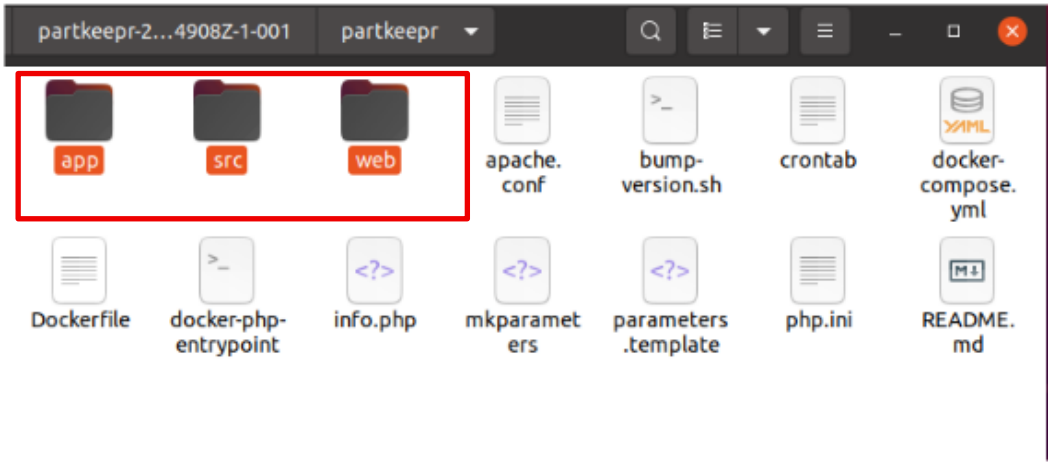
# 15 Applying the latest updates to PartKeepr

## 15.1 Objective

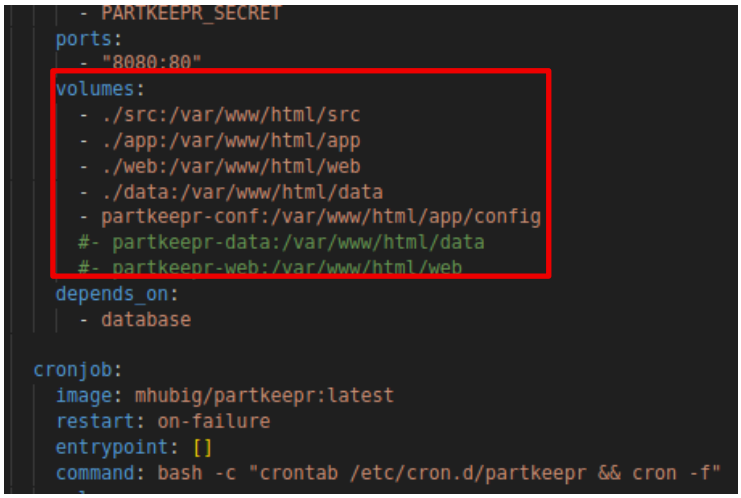
At the end of this procedure, the PartKeepr instance will include all the modifications developed in this project.

## 15.2 Replacement of folders with updated code

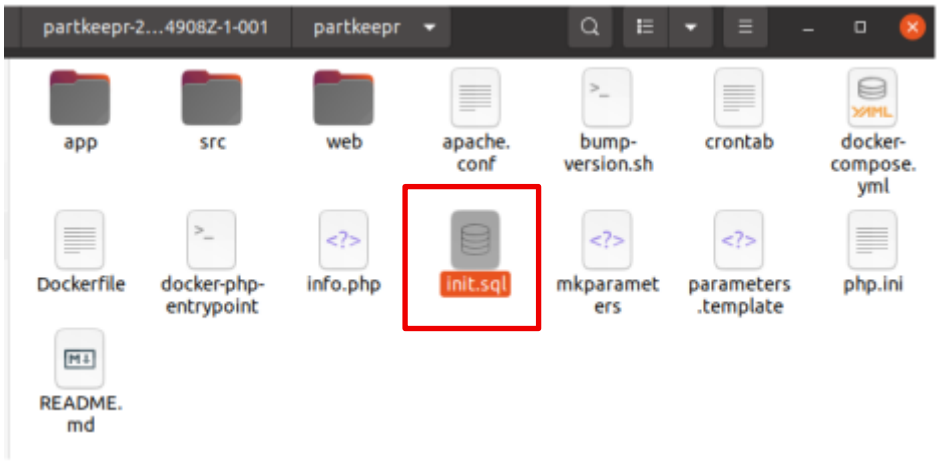
- Inside the partkeepr folder, copy the app, src and web folders from the GitHub repository.




- The next task is to adjust the docker-compose.yml file. In the section that begins with volumes, add the paths of the updated folders mentioned above and comment out partkeepr-data and partkeepr-web.



- In the [GitHub repository] there is also a file called (init.sql) Download this file and copy it into the partkeepr folder.




- At this stage, open the terminal, go to the partkeepr folder, and run the following command to start the containers in the background:

```
luis@luis-VirtualBox:~/Escritorio/Partkeepr/partkeepr_install/partkeepr$ docker-compose up -d
Creating network "partkeepr_default" with the default driver
Creating partkeepr_database_1 ... done
Creating partkeepr_partkeepr_1 ... done
Creating partkeepr_cronjob_1   ... done
luis@luis-VirtualBox:~/Escritorio/Partkeepr/partkeepr_install/partkeepr$
```

- Once the Docker container has started correctly, run this command to execute the initialisation script in PartKeepr:
    - docker exec -i partkeepr_database_1 mysql -u partkeepr -ppartkeepr partkeepr < ~/ruta/init.sql

- To finish, stop the containers and then start them again. After this, when opening PartKeepr in the browser, the application will be updated to the latest version:
    - docker compose down
    - docker compose up -d

# 16 Backup and quick restoration of PartKeepr

## 16.1 Objective

A simple script named backup_partkeepr.sh is available to create a complete recovery backup of PartKeepr.

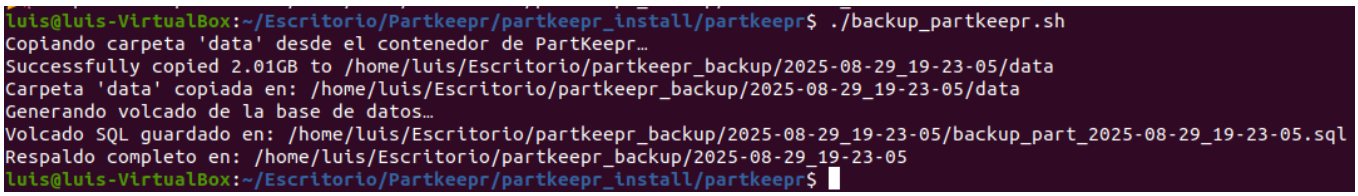
- On the one hand, it extracts the data folder from the application container, including attachments, images and temporary files managed by PartKeepr.
- On the other hand, it generates a full SQL dump of the database using mysqldump executed inside the MariaDB container.

Both artefacts are stored in a local path (by default $HOME/Desktop/partkeepr_backup) and are ready for restoration.

## 16.2 How it is executed

Place the script in any folder and launch it from a terminal with Docker running. No parameters are required if the container names match those in the file (partkeepr_partkeepr_1 and partkeepr_database_1). Otherwise, adjust the variables CONTAINER_NAME, DATABASE_CONTAINER_NAME and, if needed, BACKUP_PATH.

At the end of the process, two elements are obtained. data/ and backup_part.sql.



# 17 Github fork and update package

Following the end of maintenance of the original PartKeepr repository, an active fork has been created and published that incorporates all the improvements described in this report. To make new installations easier, the project includes in the root of the repository a folder named upgrade-kit/, designed as a self-contained update package.

The recommended workflow is deliberately short. First back up the database and the data folder with backup.sh. Then update the code and build the containers. Finally run the schema migration with init.sql.

All the material is stored in upgrade-kit/, which means there is no need to copy files manually. With this folder, the deployment and update lifecycle becomes standardised. The environment is reproducible, traceability improves since changes and migrations are versioned, and daily operation is simplified both for new installations and for instances already in production.

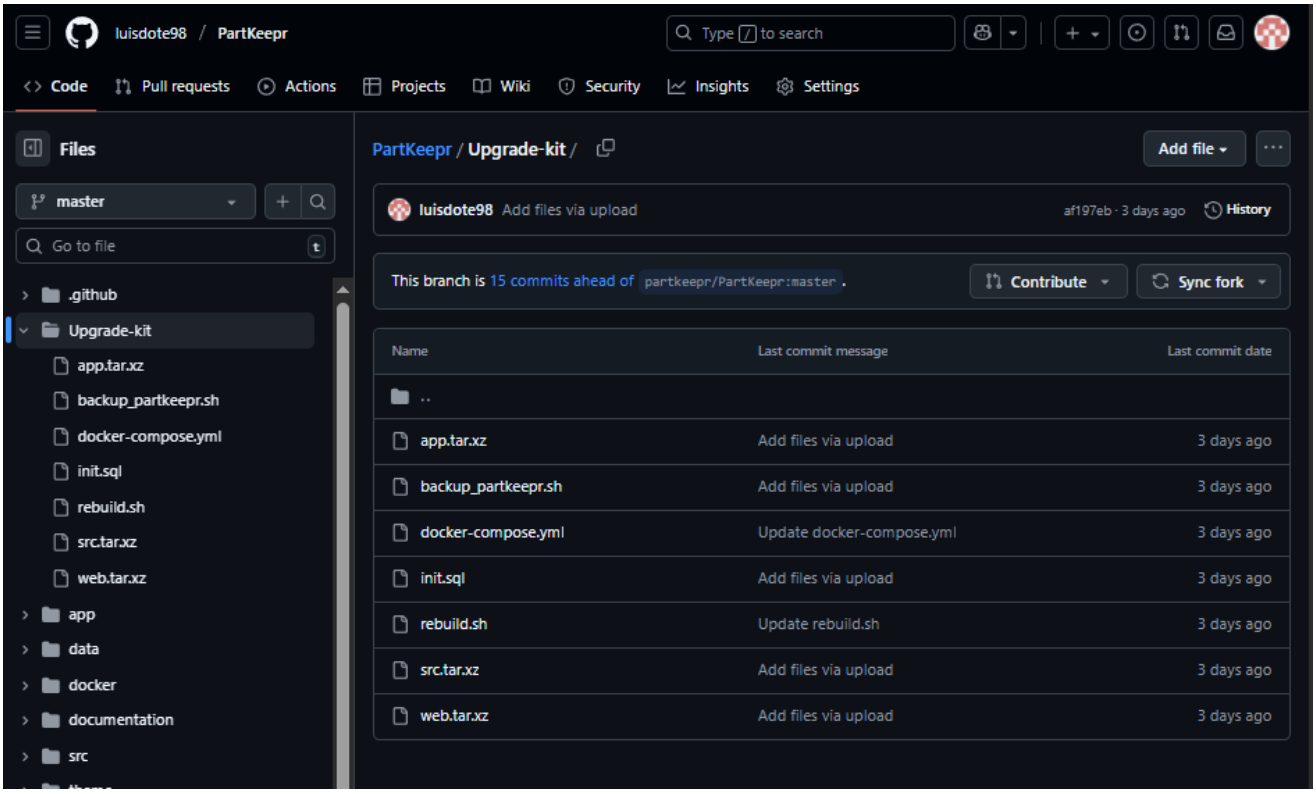



*Figure 66. Upgrade kit Github*

# 18 Conclusions

The project is considered to have achieved its main objective. PartKeepr has been left at a point where reinstalling, operating and evolving the tool no longer requires "relearning" its internals each time. The system has moved from being difficult to replicate and operationally fragile to becoming a platform with clear and repeatable procedures. In addition, the work has culminated in a real server deployment for the supervisor Andrés Roldán, from which both staff and students can use PartKeepr in their academic activity. Production use validates that the decisions taken address specific needs and that the application is stable in everyday scenarios.

## 18.1 Balance of the result

The impact can be observed on three levels. In operation, deployment and incident recovery are no longer experimental and are instead guided by proven steps such as clean installation, restoration and backups. In daily use, the application behaves more predictably. What each profile can and cannot do is more clearly defined, and navigation across large catalogues is faster. In maintenance, the architecture map and guides reduce dependency on historical memory and allow changes to be introduced with lower risk. The fact that the instance of Andrés Roldán is active and available for his students confirms the immediate transfer of these results.

## 18.2 Lessons that proved valuable

On the technical side, the experience shows that small and coherent decisions such as explicit versioning, minimal automation and validations close to the domain deliver more value than large rewrites when the scope of a project is limited. On the product side, the project confirms that targeted and well-chosen improvements such as highlighting, thumbnails, inline editing and search tools save real time and reduce errors without altering the data model. The work also required a significant exercise in reverse engineering. In the absence of official documentation, the internal structure bundles, flows, serialization and events had to be mapped by reading code and configuration in order to understand how the pieces fit together and to identify safe intervention points. This knowledge has been condensed into guides and into the published upgrade kit so that the effort invested is now reusable.

## 18.3 Limits and decisions

A major migration of the stack such as a framework upgrade or a complete change of the frontend was not attempted. The reason is not technical but strategic. With limited resources and a fixed timeframe, the priority was to secure daily operation and prepare modernisation with minimal risk. It is therefore accepted that legacy elements remain, such as language or platform, interface and libraries, which should be planned with care in the future.

## 18.4 Future gains

The main asset delivered by this work is predictability. Installation, updates, backups and restoration are no longer artisanal operations. The architecture has also been mapped in enough detail to identify where changes can be made without breaking domain invariants. This means less onboarding time for new developers, lower maintenance costs and a common ground for discussing next steps with clarity. The existence of an operational teaching instance also provides a real testbed where improvements can be validated with real users and data.

## 18.5 Closing

In practical terms, the project leaves behind an application that is safer, more usable and more manageable. The goal was not to change everything but to introduce order and method where it adds most value. With the guides, the upgrade kit, the tests and the identified extension points, the community now has a clear base to continue evolving PartKeepr without starting from scratch. Equally important, there is now a production installation that demonstrates its usefulness in a real educational environment.

# 19 Bibliography and references

[1] «Diagrama de Gantt,» [En línea]. Available: https://app.ganttpro.com/.

[2] «keepcoding,» [En línea]. Available: https://keepcoding.io/blog/salario-programador-junior-en-espana/.

[3] «Indeed - Salario programador Junio (España),» [En línea]. Available: https://es.indeed.com/career/programador-junior/salaries.

[4] «Partkeepr,» [En línea]. Available: https://github.com/partkeepr/PartKeepr.

[5] «Iventree,» [En línea]. Available: https://inventree.org/.

[6] «ERTNext,» [En línea]. Available: https://es.wikipedia.org/wiki/ERPNext.

[7] «OpenBOM,» [En línea]. Available: https://www3.technologyevaluation.com/es/solutions/61815/openbom?srsltid=AfmBOoq0QLyl_AMgBVnME0RVwqnWcCPmIarS8jzSG81rcg9IclHYgA-w.

[8] «PartsBox,» [En línea]. Available: https://partsbox.com/es/.

[9] «Symfony,» [En línea]. Available: https://symfony.com/releases/2.8.

[10] «FOSRestBundle,» [En línea]. Available: https://github.com/FriendsOfSymfony/FOSRestBundle.

[11] «DunglasApiBundle,» [En línea]. Available: https://github.com/theofidry/DunglasApiBundle.

[12] «Guia completa sobre bundles en Symfony,» [En línea]. Available: https://nelkodev.com/blog/guia-completa-sobre-bundles-en-symfony/.

[13] D. holloway, 2022. [En línea]. Available: https://es.isms.online/iso-27001/annex-a-9-access-control/.

[14] «Intersoft consulting,» [En línea]. Available: https://gdpr-info.eu/art-32-gdpr/.

[15] «cloudflare,» [En línea]. Available: https://www.cloudflare.com/es-es/learning/security/glossary/what-is-defense-in-depth/.

[16] «Akamai,» [En línea]. Available: https://www.akamai.com/es/glossary/what-is-api-security.

[17] «Octopart,» [En línea]. Available: https://octopart.com/es/pulse/p/the-nexar-story.

[18] «Support Nexar,» [En línea]. Available: https://support.nexar.com/support/solutions/articles/101000494582-nexar-playground-graphql-query-examples.

[19] «Nexar,» [En línea]. Available: https://nexar.com/api.

[20] «OpenWebinars,» [En línea]. Available: https://openwebinars.net/blog/introduccion-a-docker-desarrollo-y-despliegue-con-contenedores/.