Automatització de l'extracció de dades web mitjançant Ubuntu Server en una Raspberry Pi 5

Pau Serra Sans

Enginyeria Informàtica

Àrea: GNU/Linux

Consultor: Joaquín López Sánchez-Montañés

20/06/2024

FITXA DEL TREBALL DE FINAL DE GRAU

Títol del treball:	Automatització de l'extracció de dades web mitjançant Ubuntu Server en una Raspberry Pi 5
Nom de l'autor:	Pau Serra Sans
Nom del consultor/a:	Joaquín López Sánchez-Montañés
Nom del PRA:	Montse Serra Vizern
Data d'entrega (mm/aaaa):	06/2024
Titulació:	Enginyeria Informàtica
Àrea del Treball Final:	GNU/Linux
Idioma del treball:	Català
Paraules clau:	GNU/Linux, codi obert, web scraping, Raspberry Pi

RESUM

Aquest treball de final de grau té com a objectiu implementar un sistema automatitzat d'extracció de dades web utilitzant una Raspberry Pi 5 i de programes de codi obert. L'aplicació se centra en la necessitat de recopilar dades de manera eficient i precisa amb la finalitat de monitoritzar informació en temps real.

La metodologia seguida inclou el desglossament d'una sèrie de tasques compreses en un calendari. Les tasques que s'han dut a terme són la instal·lació i configuració d'un Ubuntu Server a la Raspberry Pi 5, el desenvolupament d'un script en Python per la extracció de dades, l'ús d'una base de dades PostgreSQL per a l'emmagatzematge, i la creació d'una API en C# per permetre l'accés i manipulació de les dades. Tots aquests components s'han encapsulat en contenidors Docker per facilitar el seu desplegament, manteniment i portabilitat.

Els resultats obtinguts mostren que aquest sistema és capaç d'extreure dades de manera ràpida i precisa. La integració de diferents tecnologies ha permès crear un sistema complet i funcional, capaç de satisfer les necessitats plantejades al començament del projecte.

En conclusió, el projecte ha assolit els seus objectius inicials, implementant un sistema automatitzat d'extracció de dades web eficient. A més, es proposen millores futures com la implementació d'un sistema de registres d'activitat i errors i d'un sistema de notificacions.

ABSTRACT

This project aims to implement an automated web data extraction system using a Raspberry Pi 5 and open-source software. The application focuses on the need to collect data efficiently and accurately to monitor information in real time.

The methodology followed includes the breakdown of a series of tasks outlined in a schedule. The tasks carried out are the installation and configuration of Ubuntu Server on the Raspberry Pi 5, the development of a Python script for data extraction, the use of a PostgreSQL database for storage, and the creation of a C# API to allow access and data manipulation. All these components have been encapsulated in Docker containers to facilitate deployment, maintenance, and portability.

The results obtained show that this system can extract data quickly and accurately. The integration of different technologies has allowed the creation of a complete and functional system capable of meeting the needs set out at the beginning of the project.

In conclusion, the project has achieved its initial objectives by implementing an efficient automated web data extraction system. Additionally, future improvements are proposed, such as implementing a logging system for activity and errors and a notification system.

ÍNDEX

ÍNDEX DE FIGURES

ÍNDEX DE TAULES

1. Introducció

1.1. Justificació i motivació

Amb l'augment de la quantitat d'informació disponible a Internet i a la web, és interessant disposar d'eines que permetin extreure i emmagatzemar dades rellevants de manera eficient.

La recopilació manual de dades pot ser tediosa i propensa a errors. Mitjançant l'automatització amb un sistema com el que es proposa, es pot millorar l'eficiència del procés, estalviant temps i recursos humans a la vegada que es minimitza la possibilitat d'errors i es garanteix la consistència de la informació obtinguda.

A més a més, amb la capacitat d'executar aquest sistema de forma programada o periòdica, es pot garantir un accés regular i en temps real a les dades de la pàgina web objectiu per mantenir la base de dades actualitzada amb la última informació disponible a la web.

Això és crucial en diversos contextos com per exemple en aquells on s'ha de disposar de les dades actualitzades al moment per tenir un avantatge competitiu i on les dades puguin ser utilitzades per realitzar anàlisis que proporcionin percepcions valuoses per a la presa de decisions.

Pel que fa les tecnologies escollides, l'script es desenvoluparà amb Python ja que és un llenguatge de programació molt potent però a l'hora senzill d'aprendre i que disposa d'una gran varietat de llibreries i un sòlid suport de la comunitat. D'altra banda, la base de dades es farà amb PostgreSQL perquè permet una gran escalabilitat acompanyada d'un bon rendiment i una comunitat activa de desenvolupadors que n'ofereixen suport i millores constants. I finalment, la API s'implementarà amb C# ja que és un llenguatge de Microsoft robust, adequat per aplicacions crítiques i segures, i que ofereix una gamma àmplia de llibreries i eines pel seu desenvolupament.

Últimament, fer el disseny d'aquest sistema amb la conteniderització de Docker serà de gran valor pel desenvolupament i desplegament de l'aplicació com a conjunt, ja que aquest software ofereix eficiència i portabilitat per gestionar dependències i entorns d'aplicacions de manera aïllada.

1.2. Objectius

Els principals objectius d'aquest projecte son els següents:

- Implementar un sistema automatitzat d'extracció de dades en un servidor Ubuntu per recopilar dades d'una pàgina web.
- Documentar adequadament tots els components i processos per facilitar la comprensió del projecte.

Els objectius parcials seran:

- Instal·lar i configurar un Ubuntu Server en una Raspberry Pi 5 per desplegar-hi el sistema.
- Crear una base de dades PostgreSQL per emmagatzemar les dades recopilades.
- Desenvolupar una API REST amb C# per facilitar la comunicació a la base de dades.
- Desenvolupar un script amb Python i Beautiful Soup per extreure les dades de la pàgina web.
- Encapsular cadascun dels components en contenidors Docker per millorar l'eficiència del projecte.
- Registrar tots els passos i accions preses durant el projecte per efectuar una bona documentació.

1.3. Enfocament i metodologia

L'enfocament i la metodologia per dur a terme aquest projecte es basen en una combinació de recerca, desenvolupament pràctic i documentació detallada. A continuació, se'n descriuen els principals aspectes.

Com a punt de partida, s'ha realitzat una investigació exhaustiva sobre el tema del projecte, incloent les tecnologies rellevants com Docker, PostgreSQL, Python, Beautiful Soup, C# i altres eines necessàries pel desenvolupament.

S'han establert clarament els objectius del projecte, amb una metodologia que inclou la creació d'un cronograma i la divisió del projecte en tasques específiques ajustades en diverses etapes i dates límit per a cada una d'elles.

Pel que fa el desenvolupament pràctic, es va començar amb la instal·lació i configuració de l'entorn de desenvolupament, així com amb el desenvolupament dels diferents

components del sistema, com la base de dades, l'script de Python i l'API en C#. Aquesta fase implica la implementació de les tasques específiques definides en el pla de treball.

Addicionalment, s'han realitzat proves extenses per assegurar-se que cada component del sistema funciona com s'espera i que hi hagi una integració adequada entre ells. S'ha detectat i solucionat qualsevol error o problema durant aquesta fase.

Finalment, s'han documentat detalladament tots els aspectes del projecte, incloent la instal·lació i configuració de l'entorn, el desenvolupament dels components, les proves realitzades i els resultats obtinguts.

1.4. Planificació i temporització

	Setmana	Activitat
1	26 febrer - 3 mar	Investigació inicial sobre el tema triat
2	4 març - 10 març	
3	11 març - 17 març	
4	18 març - 24 març	Establir objectius i metodologia de treball
5	25 març - 31 març	Elaboració del pla de treball i cronograma
6	1 abril - 7 abril	Lliurament de la PAC 1
7	8 abril - 14 abril	Instal·lació i configuració de l'entorn
8	15 abril - 21 abril	Estudi tecnològic de Docker
9	22 abril - 28 abril	Creació de la base de dades
10	29 abril - 5 maig	Lliurament de la PAC 2
11	6 maig - 12 maig	Desenvolupament de la API
12	13 maig - 19 maig	Desenvolupament del script
13	20 maig - 26 maig	Lliurament de la PAC 3
14	27 maig - 2 juny	Creació dels contenidors Docker
15	3 juny - 9 juny	Proves i depuració
16	10 juny - 16 juny	
17	17 juny - 23 juny	Finalització de la memòria
18	24 juny - 30 juny	Lliurament de la PAC 4 i vídeo

Taula 1. Planificació i temporització del projecte

1.5. Diagrama de Gantt

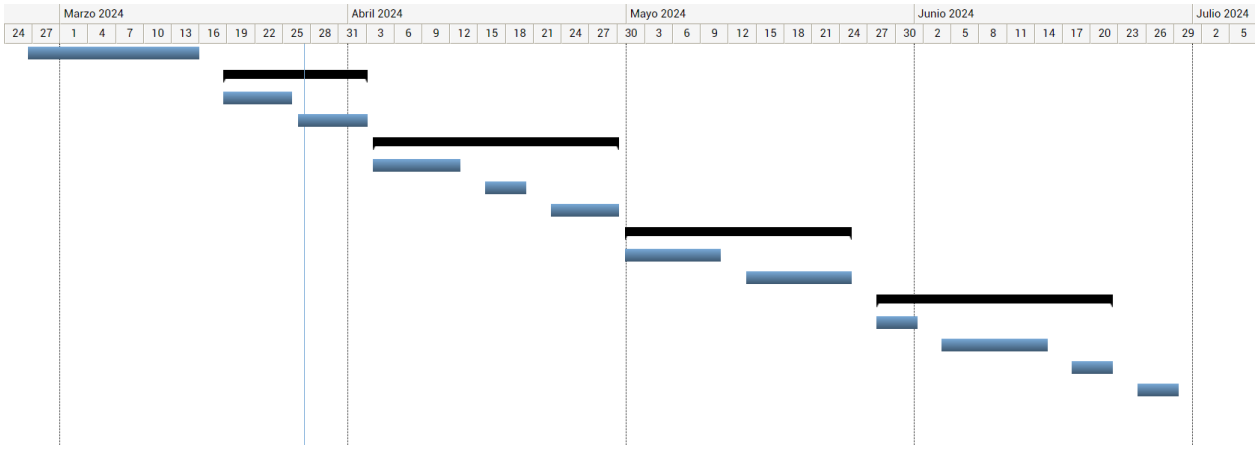

Figura 1. Diagrama de Gantt

1.6. Breu resum dels productes obtinguts

Durant aquest projecte, s'ha aconseguit implementar amb èxit un sistema complet per a l'automatització de l'extracció de dades web utilitzant una Raspberry Pi 5 i diverses tecnologies de codi obert. Els resultats principals inclouen:

- **Instal·lació i configuració del sistema operatiu**: s'ha configurat un Ubuntu Server a la Raspberry Pi per proporcionar una base sòlida per a la resta del sistema.

- **Desenvolupament d'un script de scraping**: s'ha creat un script en Python, utilitzant la biblioteca Beautiful Soup, capaç de recopilar dades automàticament des de pàgines web.

- **Emmagatzematge eficient de dades**: les dades extretes s'han emmagatzemat en una base de dades PostgreSQL de forma accessible i organitzada.

- **Creació d'una API**: s'ha desenvolupat una API REST en C# per permetre la comunicació i manipulació de les dades emmagatzemades.

- **Encapsulació en contenidors Docker**: tots els components del sistema han estat encapsulats en contenidors Docker per millorar la seva portabilitat i simplificar el desplegament i manteniment.

Aquest conjunt de resultats ha permès construir un sistema robust i funcional amb la capacitat d'executar-se de manera automàtica i eficient, i complint així els objectius

establerts al començament del projecte. Els detalls de cada fase i les tecnologies utilitzades es descriuran en profunditat en els capítols següents de la memòria.

1.7. Breu descripció dels altres capítols de la memòria

Capítol 1: Introducció
Engloba la contextualització i estudi previ per a realitzar el projecte. I seguint la metodologia exposada, també es determina la planificació i les tasques que son necessàries per assolir els objectius del projecte.

Capítol 2: GNU/Linux
S'explica el context del sistema operatiu utilitzat, s'estudien les seves característiques i beneficis, i el per què de la seva elecció.

Capítol 3: Web Scraping
Es defineix què és el web scraping, la seva història, les seves tècniques i aplicacions, així com la seva importància en l'extracció automatitzada de dades.

Capítol 4: Raspberry Pi 5
Es descriu el maquinari utilitzat per al projecte, incloent-hi les especificacions tècniques i els avantatges d'utilitzar una Raspberry Pi 5.

Capítol 5: Software
Es detalla el programari utilitzat en el projecte, incloent Ubuntu Server, Python, PostgreSQL, .NET i Docker.

Capítol 6: Implementació del sistema
Es descriu detalladament el procés pràctic d'instal·lació i configuració del sistema, desenvolupament del script, emmagatzematge de dades, creació de l'API, i l'encapsulació en Docker.

Capítol 7: Proves i anàlisi dels resultats
Es realitzen una sèrie de proves per mesurar el temps d'execució dels diferents components i s'analitza l'eficiència del sistema automatitzat.

Capítol 8: Conclusions
Es fa una valoració sobre si el que s'havia proposat a l'inici del projecte ha sigut aconseguit amb èxit i es proposen una sèrie de possibles extensions i millores futures.

2. GNU/Linux

GNU/Linux és un **sistema operatiu de codi obert** i gratuït que s'ha convertit en una opció popular per a usuaris individuals, empreses i institucions de tot el món.

Un sistema operatiu consisteix en diversos programes fonamentals que necessita l'ordinador per poder comunicar i rebre instruccions dels usuaris. Com ara llegir i escriure dades al disc dur, controlar l'ús de la memòria i executar programes. En un sistema GNU/Linux, el nucli és Linux, desenvolupat per Linus Torvalds. I la resta del sistema consisteix en altres programes, molts dels quals desenvolupats pel projecte GNU, encapçalat per Richard Stallman.

El **nucli Linux** és responsable de gestionar els recursos del sistema, com la CPU, la memòria i els dispositius de hardware. A més, proporciona les interfícies necessàries perquè altres programes puguin comunicar-se amb el maquinari de l'ordinador. Per altra banda, el **projecte GNU**, ha contribuït amb una àmplia gamma de programes que són essencials per a l'operació del sistema, com ara les utilitats de línia d'ordres, les llibreries de programació i els compiladors.

Una de les característiques principals de les distribucions GNU/Linux és la seva naturalesa oberta, modular i personalitzable. Això significa que els usuaris poden seleccionar i instal·lar només els components del sistema que necessiten per a les seves tasques específiques, des de distribucions lleugeres i optimitzades per a servidors com Ubuntu Server, fins a versions amb interfícies gràfiques com Ubuntu Desktop.

GNU/Linux també destaca per la seva naturalesa de codi obert. Això significa que la comunitat de desenvolupadors pot revisar constantment el codi font del sistema operatiu per identificar i corregir vulnerabilitats de seguretat de manera àgil. Aquest model el fa ser una plataforma estable, fiable i segura ja que els errors poden ser detectats i solucionats ràpidament per experts arreu del món.

Aquesta comunitat tendeix a ser proactiva en la identificació i resolució de problemes de seguretat, amb actualitzacions i correccions disponibles amb regularitat per mantenir els sistemes protegits contra amenaces. Tot això, converteix a GNU/Linux en una opció molt atractiva per a entorns on la seguretat és una prioritat elevada.

3. Web scraping

3.1. Què és?

El web scraping és una tècnica que s'utilitza per a **extreure dades** o informació d'un lloc web. Aquesta tècnica implica l'ús de programari per a analitzar el contingut d'una pàgina web i extreure les dades rellevants d'acord amb uns criteris predefinits.

3.2. Història

Amb el naixement i expansió de la World Wide Web durant la dècada dels 1990, els motors de cerca començaven a emergir com a portals d'accés a la vasta quantitat d'informació en línia. Però aquests motors de cerca necessitaven una manera d'extreure informació de la web i indexar-la perquè els usuaris poguessin trobar-la fàcilment.

És llavors quan apareix el concepte de *web scraping*. En un principi, aquesta pràctica es basava en tècniques relativament simples, com en programes que navegaven per les pàgines web i extreien la informació mitjançant l'anàlisi del codi font HTML.

Posteriorment, a principis dels anys 2000 es va veure un important creixement en la pràctica del web scraping, així com en el desenvolupament d'eines i tecnologies per a aquest propòsit. Un dels esdeveniments clau va ser l'aparició de Python, que va guanyar popularitat gràcies a les seves biblioteques i frameworks específics pel scraping, com Beautiful Soup. Aquestes eines van simplificar significativament el procés d'extracció de dades, permetent als desenvolupadors extreure la informació rellevant de manera estructurada i fàcilment manipulable.

Paral·lelament, els avanços en tecnologia i connectivitat també van contribuir a l'expansió d'aquesta tècnica. L'augment de la velocitat de connexió a Internet i la disponibilitat de servidors més potents van permetre processar i analitzar grans volums de dades amb més rapidesa i eficiència.

Durant els anys següents, una de les principals innovacions va ser el scraping basat en el navegador. Això va permetre als usuaris simular la interacció humana amb les pàgines mitjançant un navegador web. Selenium, una de les eines més populars, permet als usuaris automatitzar el navegador per interactuar amb el contingut web, fent clics, omplint formularis i fins i tot interactuant amb elements dinàmics com menús desplegables.

Aquest nou enfocament va obrir noves oportunitats per a la recopilació de dades, com la captura de contingut generat dinàmicament i la automatització de tasques de navegació web. Tot i això, aquesta tècnica també presenta alguns desavantatges. Per exemple, pot ser més lent i consumir més recursos que el scraping tradicional, i pot ser més difícil de mantenir a mesura que els llocs web canvien el seu disseny i la seva estructura.

A mesura que el web scraping s'ha anat convertint en una pràctica més comuna i sofisticada, han sorgit debats entorn a la seva legalitat. Molts llocs web inclouen termes de servei que prohibeixen explícitament l'ús de robots o altres tècniques de web scraping per recopilar dades del seu contingut. A més s'han establert normatives i regulacions, especialment en el context de la protecció de dades personals i la privadesa.

Avui en dia, el web scraping s'utilitza en una àmplia gamma d'aplicacions i industries. Una de les aplicacions més comunes és en l'àmbit del comerç electrònic, on es fa servir per a monitoritzar preus, recopilar ressenyes de productes i analitzar tendències del mercat.

3.3. Tècniques

Com ja s'ha comentat, web scraping és el procés de recopilar informació de forma automàtica del web. Per a dur a terme aquesta pràctica, es disposa de vàries solucions amb diferents nivells d'automatització que les existents tecnologies de web scraping poden oferir:

- **Copiar i enganxar humà**: és la pràctica per extreure manualment dades d'una pàgina web mitjançant les funcions de copiar i enganxar del navegador web. Pot ser útil per a tasques simples i ocasionals de recopilació de dades, però no és una solució eficient per a extraccions de dades a gran escala o processos repetitius.

- **Scraping basat en API**: alguns llocs web proporcionen una API que permet als desenvolupadors accedir a les seves dades de manera estructurada i programàtica. Utilitzar una API és una opció que ofereix diversos avantatges, com una millor estructura de dades, que són fàcilment accessibles i estan dissenyades per a la integració amb altres aplicacions i, per últim, més fiabilitat i conformitat amb els termes del servei del lloc web.

- **Scraping per automatització del navegador**: quan una pàgina web utilitza JavaScript per generar o modificar el seu contingut de manera dinàmica, l'scraping convencional pot no ser suficient. En aquests casos, l'automatització del navegador permet controlar un navegador web (Chrome, Firefox, Edge, etc.) de forma programàtica.

- **Llibreries de web scraping**: aquesta tècnica es basa en l'ús d'eines per analitzar i extreure dades del codi HTML d'una pàgina web. La primera etapa consisteix en obtenir el codi font HTML de la pàgina web que es vol analitzar. Un cop descarregada la pàgina, mitjançant tècniques de parseig del HTML, s'analitza el codi per identificar els elements que contenen les dades que es vol extreure. Utilitzant seleccions per etiqueta, classe o identificador s'extreuen les dades contingudes i finalment es processen per estructurar-les, netejar-les o emmagatzemar-les.

Aquesta última tècnica serà la emprada en aquest projecte. S'utilitzaran les llibreries *Requests* i *Beautiful Soup* de Python per obtenir i manipular el codi HTML rellevant.

4. Raspberry Pi 5

La Raspberry Pi és un ordinador de placa única **SBC** (*Single-Board Computer*) desenvolupat al Regne Unit per la Fundació Raspberry Pi amb associació de Broadcom.

Aquest tipus de dispositius proporcionen una sèrie d'avantatges amb un gran atractiu:

- Un **menor consum elèctric**, aquests sistemes acostumen i poden estar sempre encesos amb una capacitat de dissipació calorífica mínima, el que suposa un menor consum i un estalvi econòmic important.

- Una **mida i preu reduïts**, es tracta de ordinadors petits i barats que, tot i això, mantenen un gran potència i gamma de prestacions.

- Funcionen amb **sistemes operatius lliures**, la majoria utilitzen versions lliures de Linux adaptades als processadors d'aquests sistemes (ARM).

En concret, la Raspberry Pi 5 va ser anunciada el setembre del 2023 amb millores del hardware i software que la fan el doble o triple de potent que la seva antecessora, la Pi 4.

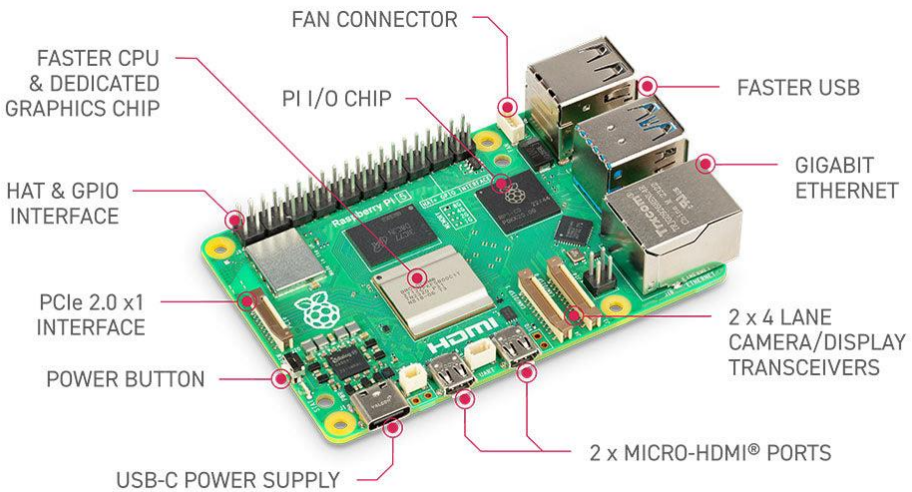

Figura 2. Esquema amb especificacions de la Raspberry Pi 5

En la següent taula podem observar amb detall les seves característiques principals.

Processador	Broadcom BCM2712 2.4GHz quad-core 64-bit Arm Cortex-A76 CPU, amb Extensió Criptogràfica, 512KB per nucli L2 cau, i una 2MB compartida L3 cau.
Característiques	• GPU VideoCore VII, suportant OpenGL ES 3.1, Vulkan 1.2 • Doble 4Kp60 HDMI display output amb suport HDR • Descodificador 4Kp60 HEVC • LPDDR4X-4267 SDRAM (4GB o 8GB) • Dual-band 802.11ac Wi-Fi • Bluetooth 5.0 / Bluetooth Low Energy (BLE) • Ranura per tarjeta microSD, suport mode high-speed SDR104 • 2 × USB 3.0 ports • 2 × USB 2.0 ports • Gigabit Ethernet • 2 × 4-lane MIPI transceptors càmera/display • Interfície PCIe 2.0 x1 • Transformador USB-C 5V/5A DC • Capçalera de 40 pins • Rellotge en temps real (RTC) • Botó d'encendre

Taula 2. Característiques Raspberry Pi 5

5. Software

5.1. Ubuntu Server

Ubuntu Server és una distribució de sistema operatiu de codi obert basada en l'arquitectura de **Debian**, cosa que significa que el seu codi font és lliure per ser descarregat, utilitzat i modificat per qualsevol persona.

A diferència de la versió d'escriptori de Ubuntu, que està destinada principalment a usuaris finals, Ubuntu Server està especialment optimitzada per a entorns de servidor, oferint una plataforma robusta i fiable per a diverses aplicacions i serveis.

Una de les característiques principals de Ubuntu Server és la seva estabilitat i seguretat. Els desenvolupadors ofereixen actualitzacions regulars per mantenir el sistema protegit contra amenaces de seguretat i per optimitzar el rendiment. Això ajuda a garantir que els servidors que utilitzen Ubuntu Server siguin fiables i estables.

Ubuntu Server també destaca per la seva facilitat d'ús i la seva àmplia gamma d'eines i recursos per simplificar la configuració i la gestió del servidor. Això inclou eines de configuració com ara la *command-line interface* (CLI).

5.2. Python

Python és un llenguatge de programació interpretat, de codi obert que va ser creat per Guido van Rossum i publicat per primera vegada el 1991. És famós per la seva simplicitat i llegibilitat del codi, el que el fa popular tant entre principiants com entre desenvolupadors experimentats.

Una de les característiques clau de Python és la seva sintaxi clara i elegant, que permet als desenvolupadors expressar conceptes de manera concisa i eficient. Aquest llenguatge és versàtil i pot ser utilitzat en una àmplia gamma d'àrees, com ara desenvolupament web, anàlisi de dades, intel·ligència artificial i ciberseguretat, entre d'altres.

Python és recolzat per la seva gran comunitat de desenvolupadors actius i el suport d'una àmplia gamma de biblioteques i frameworks que faciliten el desenvolupament de diverses aplicacions. Algunes de les biblioteques més populars inclouen NumPy i Pandas per a l'anàlisi de dades, TensorFlow i PyTorch per a la intel·ligència artificial i Django i Flask

per al desenvolupament web, per exemple. En el cas del web scraping, les biblioteques més populars de Python són Beautiful Soup, Selenium, Requests i Scrapy.

5.3. PostgreSQL

PostgreSQL és un potent sistema de bases de dades relacional de codi obert que utilitza i amplia el llenguatge SQL combinat amb moltes funcionalitats que emmagatzemen i escalen de manera segura les càrregues de treball de dades més complicades.

PostgreSQL ve amb moltes funcionalitats destinades a ajudar els desenvolupadors a construir aplicacions, els administradors a protegir la integritat de les dades i a construir entorns tolerants a falles, i a ajudar a gestionar les dades, ja siguin petites o grans. A més, PostgreSQL és altament extensible. Per exemple, es poden definir els tipus de dades, desenvolupar funcions personalitzades i fins i tot escriure codi en diferents llenguatges de programació sense haver de recompilar la base de dades.

5.4. C#

C# és un llenguatge de programació modern, orientat a objectes i fortament tipat. Va ser desenvolupat per Microsoft i és el més utilitzat i destacat dins de la seva iniciativa .NET.

5.4.1. Característiques

C# és conegut per les seves nombroses característiques que el fan destacar com un llenguatge de programació eficient. Una de les característiques més destacades és la orientació a objectes, la qual permet als desenvolupadors crear aplicacions modulars i reutilitzables.

A més, C# garanteix la seguretat de tipus, ajudant a prevenir errors relacionats amb els tipus de dades. Cada variable i constant en C# té un tipus específic que no pot ser canviat, i aquesta seguretat de tipus es comprova en temps de compilació per evitar errors.

Una altra característica important de C# és la gestió automàtica de la memòria. El llenguatge inclou un recol·lector d'elements no utilitzats automàtic que gestiona la memòria de forma eficient.

5.4.2. Aplicacions

C# és un llenguatge versàtil que s'utilitza en diverses àrees del desenvolupament de programari:

- **Aplicacions de Windows**: C# és àmpliament utilitzat per desenvolupar aplicacions d'escriptori per a Windows amb eines com Windows Forms i Windows Presentation Foundation (WPF).

- **Aplicacions web**: Amb ASP.NET, C# es converteix en una opció popular per al desenvolupament d'aplicacions web dinàmiques i serveis web, oferint una infraestructura sòlida i flexible per a projectes web.

- **Aplicacions mòbils**: Xamarin, una plataforma basada en C#, permet als desenvolupadors crear aplicacions mòbils per a Android i iOS utilitzant una base de codi compartida, la qual cosa redueix el temps i l'esforç necessaris per desenvolupar aplicacions multiplataforma.

- **Jocs**: C# és el llenguatge principal utilitzat en el desenvolupament de jocs amb Unity, un motor de jocs popular que permet crear jocs per a diverses plataformes que ofereix una experiència de desenvolupament potent i flexible.

5.4.3. .NET

Com s'ha comentat, C# forma part de .NET. És tracta d'una plataforma de desenvolupament de codi obert, multiplataforma i gratuïta de Microsoft, dissenyada per compilar una gran varietat d'aplicacions.

És compatible amb múltiples llenguatges i proporciona un entorn d'execució d'alt rendiment utilitzat en aplicacions a gran escala. Addicionalment, .NET ofereix productivitat, seguretat, i administració automàtica de memòria.

Els components principals de .NET inclouen l'entorn d'execució, biblioteques que proporcionen funcionalitats, el compilador que transforma codi font en executable, l'SDK i eines per crear i supervisar aplicacions, i les piles d'aplicacions com ASP.NET Core i Windows Forms per desenvolupar aplicacions.

5.5. Docker

Docker és una plataforma de codi obert dissenyada per automatitzar el desplegament d'aplicacions en **contenidors**. Els contenidors són unitats lleugeres, portàtils i autosuficients que inclouen tot el necessari per executar una aplicació. Com el codi, les biblioteques, les eines del sistema i els paràmetres de configuració.

Com que els contenidors comparteixen el mateix nucli del sistema operatiu de l'amfitrió, fan un ús més eficient dels recursos, a diferència de les màquines virtuals tradicionals. Aquesta tecnologia permet crear, desplegar i executar aplicacions de manera consistent independentment de l'entorn en què s'executin.

El principal avantatge de Docker és la capacitat d'aïllar les aplicacions i les seves dependències en contenidors independents, assegurant així que el funcionament d'un contenidor no afecti els altres. Això simplifica substancialment la gestió de dependències i configuracions, reduint els conflictes entre diferents entorns de desenvolupament i producció.

Les principals característiques de Docker són:

- **Portabilitat**: els contenidors Docker poden executar-se en qualsevol sistema que tingui el motor de Docker instal·lat, ja sigui una màquina local, un servidor al núvol o un entorn híbrid.

- **Consistència**: en empaquetar una aplicació i les seves dependències dins d'un contenidor, s'assegura que el codi s'executi de la mateixa manera en desenvolupament, proves i producció.

- **Escalabilitat**: Docker permet desplegar i gestionar fàcilment múltiples instàncies d'una aplicació per facilitar l'escalabilitat horitzontal.

- **Seguretat**: els contenidors proporcionen un cert nivell d'aïllament entre aplicacions per millorar la seguretat i minimitzar l'impacte d'una potencial vulnerabilitat.

5.5.1. Docker Compose

Docker Compose és una eina que permet definir i executar aplicacions **multi-contenidor**. Mitjançant un **fitxer YAML**, es poden especificar els serveis que conformen l'aplicació, les seves dependències, volums i xarxes, fent així més fàcil la gestió dels contenidors.

Un fitxer *docker-compose.yaml* descriu la configuració d'una aplicació en diversos serveis. Per exemple, una aplicació web pot requerir un servidor web, una base de dades i una API, cadascun executant-se en contenidors separats però interconnectats.

L'ús de Docker Compose ofereix diversos avantatges que simplifiquen el desenvolupament, el desplegament i la gestió d'aplicacions contenidoritzades:

- **Control simplificat**: Docker Compose permet definir i gestionar aplicacions multi-contenidor en un únic fitxer YAML. Això simplifica la tasca de coordinar diversos serveis.

- **Desenvolupament ràpid d'aplicacions**: la configuració utilitzada per crear un contenidor s'emmagatzema a la memòria cau. Quan es reinicia un servei que no ha canviat, Docker Compose reutilitza els contenidors existents. Això permet fer canvis a l'entorn de manera molt ràpida.

- **Portabilitat**: s'admet l'ús de variables en un fitxer de Docker Compose. Es poden utilitzar aquestes variables per personalitzar la composició per a diferents entorns o diferents usuaris.

- **Comunitat i suport extensos**: Docker Compose es beneficia d'una comunitat activa que proporciona recursos abundants, manuals i suport. Aquest ecosistema impulsat per la comunitat contribueix a la millora contínua de Docker Compose i ajuda els usuaris a resoldre problemes de manera efectiva.

5.5.2. Imatges

Les imatges de Docker són **plantilles** de només lectura que contenen instruccions per a crear un contenidor. Una imatge és una instantània o un esquema de les biblioteques, dependències i arxius necessaris dintre d'un contenidor per a que s'executi una aplicació.

Les imatges són immutables, el que significa que no es poden modificar una vegada creades, però si que es poden duplicar, compartir o eliminar. Aquestes, es creen a partir d'un *Dockerfile,* un arxiu de text que conté les instruccions per crear la imatge.

Les imatges es construeixen en capes, on cada instrucció del *Dockerfile* crea una nova capa permetent la reutilització de capes entre diferents imatges, reduint la mida i accelerant els desplegaments.

Crear una imatge personalitzada implica escriure un *Dockerfile* des de zero que contingui les instruccions que un usuari podria cridar a la línia de comandaments. En aquesta taula queden reflectides totes les instruccions que es poden utilitzar al crear una imatge:

Instrucció	Descripció
ADD	Afegir fitxers i directoris locals o remots.
ARG	Utilitzar variables en temps de construcció.
CMD	Especificar comandes.
COPY	Copiar fitxers i directoris.
ENTRYPOINT	Especificar l'executable per defecte.
ENV	Establir variables d'entorn.
EXPOSE	Descriure quins ports escolta l'aplicació.
FROM	Crear una nova capa a partir d'una imatge base.
HEATLHCHECK	Comprovar l'estat d'un contenidor en arrencar.
LABEL	Afegir metadata a una imatge.
MAINTAINER	Especificar l'autor d'una imatge.
ONBUILD	Especificar instruccions per quan la imatge s'utilitza en una construcció.
RUN	Executar comandes de construcció.
SHELL	Establir la shell per defecte d'una imatge.
STOPSIGNAL	Especificar el senyal del sistema per sortir d'un contenidor.
USER	Establir l'ID d'usuari i grup.
VOLUME	Crear volums.
WORKDIR	Canviar el directori de treball.

Taula 3. Instruccions Dockerfile

5.5.3. Docker Hub

Docker Hub és una plataforma en el núvol que proporciona serveis per a crear, emmagatzemar i distribuir imatges de Docker. Per tant, funciona com un registre públic on Docker o els mateixos usuaris poden pujar i guardar les seves imatges de contenidors.

Una de les funcionalitats de Docker Hub és la possibilitat de cercar i descarregar aquestes imatges. S'ofereixen una gran varietat d'imatges tant oficials com creades per la comunitat que cobreixen una àmplia gamma d'aplicacions i serveis, com bases de dades, servidors web, intel·ligència artificial i sistemes operatius, entre d'altres.

A més, per controlar la gestió de les diferents versions d'una imatge, els repositoris poden arribar a tenir múltiples etiquetes, on cadascuna pot representar una versió, un estat o una configuració específica de la imatge.

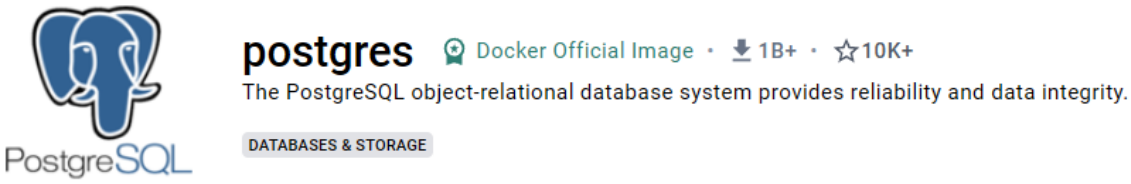

Figura 3. Exemple d'imatge pujada a Docker Hub

6. Implementació del sistema

En aquest apartat s'explicarà pas per pas el procés pràctic que s'ha dut a terme per a per implementar el sistema complet juntament amb els programes utilitzats per fer-ho.

6.1. Instal·lació de Ubuntu Server

En aquest projecte es va començar utilitzant la versió 23.10 de Ubuntu Server (*Mantic Minotaur*). Això es degut a que la Raspberry Pi 5, en el moment de la instal·lació del sistema operatiu, no suportava la versió LTS (*Long-term Support*) actual 22.04 (*Jammy Jellyfish*), i la següent versió LTS 24.04 (*Noble Numbat*) encara no estava disponible.

Tanmateix, durant el transcurs del projecte, Ubuntu ha publicat la versió LTS 24.04 i s'ha optat per actualitzar el sistema operatiu ja que la versió instal·lada 23.10 va EOL (*End Of Life*) i quedarà sense suport.

6.1.1. Raspberry Pi Imager

Per a començar amb la instal·lació de Ubuntu Server 23.10 s'ha utilitzat una eina de programari desenvolupada per la Fundació Raspberry Pi que simplifica el procés d'instal·lació de diferents sistemes operatius en una Raspberry Pi, ja que permet seleccionar, descarregar i escriure la imatge del sistema operatiu directament a la targeta SD de manera guiada.

Figura 4. Pantalla d'inici de Raspberry Pi Imager

Una vegada seleccionat el dispositiu, sistema operatiu i emmagatzematge, aquesta eina també permet preconfigurar el sistema operatiu abans de la primera arrencada, personalitzant paràmetres com usuari i contrasenya, Wi-Fi, *hostname*, etc., i serveis com SSH.

En el següent manual oficial queda explicat tot el procediment en més detall:

https://www.raspberrypi.com/documentation/computers/getting-started.html#install-using-imager

6.1.2. Configuració de Ubuntu Server

Quan ja es disposa del sistema operatiu escrit a la targeta SD, s'ha comprovat si aquest necessita alguna actualització. I, per altra banda, s'ha configurat la xarxa. En Ubuntu 23.10 la configuració de la xarxa es dur a terme amb la utilitat **Netplan**, que és una descripció YAML de les interfícies de xarxa requerides.

En el nostre projecte, s'ha assignat al servidor una IP estàtica de la xarxa local a la interfície de *ethernet*. També se li han especificat els servidors DNS de Cloudfare i Google, ja que són els més ràpids, i la passarel·la (*gateway*) de l'enrutador de la xarxa local.

```
network:
  version: 2
  ethernets:
    eth0:
      addresses:
        - 192.168.1.15/24
      nameservers:
        addresses:
          - 1.1.1.1
          - 8.8.8.8
          - 1.0.0.1
          - 8.8.4.4
      routes:
        - to: default
          via: 192.168.1.1
```

Figura 5. Configuració Netplan del servidor

6.2. Recopilació de dades mitjançant Python

Python és un dels llenguatges de programació més populars per a la recopilació de dades web gràcies a la seva simplicitat i a la gran quantitat de biblioteques disponibles que en faciliten la tasca.

6.2.1. Visual Studio Code

L'editor que s'ha utilitzat per a realitzar el desenvolupament d'aquest component del projecte és el Visual Studio Code. El Visual Studio Code és un editor de codi font desenvolupat per Microsoft lleuger però potent que s'executa a l'escriptori i està disponible per a Windows, macOS i Linux.

VS Code presenta una interfície d'usuari neta i intuïtiva que facilita la navegació i la manipulació del codi. Una de les característiques més destacades de VS Code és la seva capacitat d'extensió, ja que disposa d'un vast mercat d'extensions que permeten als usuaris afegir funcionalitats addicionals segons les seves necessitats. Això inclou extensions per a llenguatges de programació (C++, C#, Java, Python, PHP, .NET), integració amb sistemes de control de versions com Git, depuradors i eines de desenvolupament web, entre d'altres.

6.2.2. Desenvolupament del script

Pel desenvolupament d'aquest script s'han utilitzat les següents biblioteques principals:

- **Requests**: permet fer peticions HTTP de manera senzilla i elegant.

- **Beautiful Soup**: permet extreure informació de fitxers HTML o XML. Fent ús d'eines d'anàlisi d'aquests llenguatges, proporciona funcions per navegar, cercar i modificar l'arbre sintàctic.

També s'han utilitzat biblioteques per simplificar processos. **Datetime** i **Time** per manipular dates, hores i temps, i **Regular Expressions (re)** per cercar i manipular text basat en patrons.

Pel que fa l'estructura del codi del script, està compresa en una funció principal que inicia un bucle *while* que s'executa de forma indefinida i que realitza les següents tasques en cada iteració:

Es fa una petició GET amb l'ús de la biblioteca *Requests* a la pàgina web objectiu i se n'obté la resposta.

```
get_response = requests.get(URL)
```

En cas que aquesta petició retorni un codi d'estat HTTP 200 OK, volent dir que s'ha pogut aconseguir correctament el codi font, es continua amb el procés d'obtenció de les dades. En cas que el codi d'estat sigui diferent, voldrà dir que la petició no és vàlida i que no s'ha pogut obtenir el codi HTML, per tant, es mostra un missatge d'error.

Si la petició és vàlida, s'utilitza la biblioteca Beautiful Soup per analitzar el contingut HTML de la pàgina i, amb la funció *find_all*, es cerquen, filtren i guarden en una variable tots els elements que tenen la informació rellevant per classe.

```
if get_response.status_code == 200:
   html_soup = BeautifulSoup(get_response.text, 'html.parser')
   data = html_soup.find_all('element', class_='exemple')
```

La funció *find_all* retorna un llistat format pels elements que s'ha indicat. Cada element, es manipula per eliminar les línies buides o que no son rellevants i, posteriorment, es converteix per ser emmagatzemat a un diccionari.

Cada diccionari de dades és enviat a la API fent una petició POST amb l'ús de la biblioteca Requests en format JSON.

```
requests.post(url="http://IP_API", json=diccionari)
```

Exemple del JSON que s'envia a la API:

```
{
    "ItemNumber": "75686-1",
    "Name": "Lego Test",
    "Parts": 654,
    "Theme": "Star Wars",
    "YearReleased": 2024
}
```

Finalment, abans de tornar a l'inici del bucle, es fa una pausa d'espera d'uns minuts establerts.

```
time.sleep(SLEEP_MINUTES * 60)
```

6.3. Emmagatzematge de dades en PostgreSQL

PostgreSQL és un sistema de gestió de bases de dades versàtil i fiable. Destaca pel seu bon rendiment i funcionalitat. La seva arquitectura d'emmagatzematge permet una gestió eficient de les dades, fent-lo adequat per a una àmplia gamma d'aplicacions, des de petites fins a sistemes empresarials de gran escala.

6.3.1. DBeaver

DBeaver és una eina de bases de dades que s'utilitzarà en aquest projecte per poder depurar i veure gràficament l'estructura de la base de dades. DBeaver és una aplicació de codi obert i multiplataforma, dissenyada per proporcionar una interfície d'usuari intuïtiva i funcional que facilita la gestió i la interacció amb bases de dades.

DBeaver suporta una àmplia varietat de bases de dades, tant SQL com NoSQL, incloent MySQL, PostgreSQL, SQLite, Oracle, SQL Server, MongoDB, entre d'altres. També ofereix una gran quantitat de funcionalitats avançades, com l'editor de dades, l'editor de SQL, esquemes de bases de dades, diagrames ER (*Entity-Relationship*), exportació i importació de dades i generació de dades fictícies.

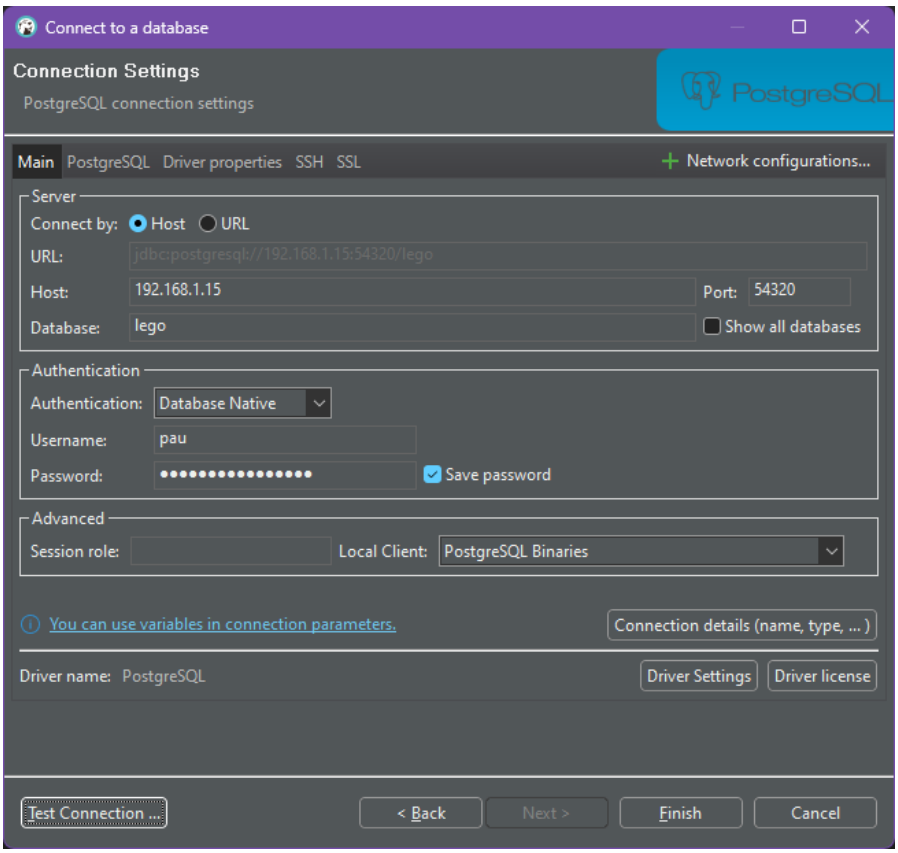

Figura 6. Pantalla d'accés a una base de dades amb DBeaver

6.4. Desenvolupament de la API en C#

L'API (*Application Programming Interface*) és una aplicació que no es directament accessible per l'usuari i encarregada de manipular les dades (*backend*) construïda en C# amb el framework .NET per gestionar operacions relacionades amb col·leccions obtingudes. Utilitza Entity Framework Core per interactuar amb una base de dades PostgreSQL, permetent operacions CRUD (Create, Read, Update, Delete) sobre entitats.

6.4.1. Visual Studio 2022

L'entorn de desenvolupament que s'ha utilitzat per a crear la API en C# és el Visual Studio 2022. És la versió més moderna d'aquest IDE de Microsoft que incorpora millores i novetats respecte la seves versions anteriors.

A part de haver estat redissenyat amb una interfície més neta i moderna, una de les millores més destacades de Visual Studio 2022 és la seva capacitat per gestionar projectes més grans i complexos amb un major rendiment.

A més, C#, entre altres llenguatges, és compatible amb IntelliCode. Intellicode és una eina d'assistència de codi impulsada per intel·ligència artificial desenvolupada per Microsoft per a Visual Studio 2022. Aquesta eina, ofereix suggeriments de codi més precisos i rellevants adaptats al context del projecte i ajuda a revisar i identificar errors o inconsistències.

Visual Studio també ofereix depuració del codi. La depuració és el procés d'identificar i corregir errors en el codi utilitzant eines com els punts de ruptura, que permeten posar pausa en l'execució per inspeccionar l'estat del programa i l'execució pas a pas per comprendre el comportament del codi.

6.4.2. Dependències

Pel desenvolupament de la API s'ha necessitat importar dos paquets que s'inclouen a les dependències del projecte.

- *Microsoft.EntityFrameworkCore*: el qual permet treballar amb bases de dades utilitzant objectes .NET, fent així més senzill el codi per accedir a les dades. Per exemple, permet crear models mitjançant classes, gestionar els canvis d'esquema de la base de dades amb migracions i fer consultes utilitzant LINQ.

- *Npgsql.EntityFrameworkCore.PostgreSQL*: aquest paquet permet a EF Core interactuar amb una base de dades PostgreSQL, traduint les consultes LINQ a consultes SQL que PostgreSQL pot executar.

6.4.3. Configuració de l'aplicació

El fitxer *appsettings.json* conté la configuració necessària per connectar-se a la base de dades creada anteriorment. Aquesta configuració es fa en format *connection string* (cadena de connexió), i s'utilitza per especificar com l'aplicació s'ha de connectar a la base de dades o a altres serveis que necessitin autentificació.

La cadena de connexió inclou paràmetres com l'adreça del servidor, el port, l'usuari, la contrasenya, el nom de la base de dades i altres configuracions. La cadena que s'utilitzarà en aquesta aplicació és la següent:

```
"ConnectionStrings": {
  "name": "User Id=user;Password=password;Server=ip_bbdd;Port=5432;Database=database;"
}
```

Name fa referència a l'identificador que després s'utilitzarà per obtenir la cadena de connexió; *user*, a l'usuari creat per accedir a la base de dades; *password*, la contrasenya d'aquest usuari; *ip_bbdd*, l'adreça on està ubicada la base de dades; i *database*, el nom de la base de dades.

6.4.4. Model

Un model és una representació estructurada de les dades que l'API maneja i intercanvia amb els clients i s'utilitzen en els controladors per rebre i retornar dades.

En un model es defineix una classe que conté **propietats** que representen els camps de la base de dades. Les propietats són variables que tenen un tipus de dades associat (*string*, *int*, *DateTime*, etc.). Per assegurar que les dades compleixin amb certs requisits, es poden utilitzar opcionalment atributs de validació com [Required], [StringLength], [Key], entre d'altres. Aquests atributs (*Data Annotations*), ajuden a validar les dades automàticament abans de processar-les.

En el cas del nostre projecte, s'utilitza aquest model per definir un set de Lego:

```
public class LegoSet
{
    [Key]
    public required string ItemNumber { get; set; }

    [Required]
    public required string Name { get; set; }

    [Required]
    public required int Parts { get; set; }

    [Required]
    public required string Theme { get; set; }

    [Required]
    public required int YearReleased { get; set; }
}
```

En aquest, queden definits els camps creats a la base de dades. En el cas de *ItemNumber*, s'especifica que es la clau primària i en els altres camps, que són obligatoris.

6.4.5. Context

Aquest fitxer defineix el context de la base de dades. Hereta de *DbContext* i conté un *DbSet* per a cada model que es vol gestionar amb EF Core.

El *DbContext* és una classe que representa una sessió amb la base de dades. Manté un seguiment dels objectes que estan relacionats amb la base de dades i coordina les operacions CRUD en aquests objectes. El *DbContext* s'utilitza per configurar la connexió a la base de dades, mapejar les classes a les taules de la base de dades i realitzar operacions de consulta i modificació en aquestes dades.

Per altra banda, el *DbSet* és una propietat de *DbContext* que representa una col·lecció d'entitats d'un tipus específic en el context de la base de dades. Cada *DbSet* correspon a una taula a la base de dades i proporciona mètodes per realitzar operacions a la taula associada.

El context de la nostre aplicació contindrà un *DbSet* de col·lecció del model definit com a *Sets*.

```
public class AppDbContext(DbContextOptions<AppDbContext> options) : DbContext(options)
{
    public DbSet<LegoSet> Sets { get; set; }
}
```

6.4.6. Controlador

Un controlador és una classe que s'encarrega de gestionar les **sol·licituds HTTP** i proporcionar les respostes adequades. Aquestes sol·licituds poden ser de diversos tipus (GET, POST, PUT, DELETE, etc.) i el controlador és responsable de processar-les, interactuar amb la lògica i retornar les dades o resultats adequats.

Els controladors s'acompanyen amb atributs com [ApiController], que indica que la classe és un controlador d'API i habilita comportaments específics com la validació automàtica dels models, i l'atribut [Route], que defineix la ruta de les sol·licituds HTTP que el controlador ha de manejar.

A part, els mètodes públics del controlador, anomenats *endpoints*, estan marcats amb atributs com [HttpGet], [HttpPost], [HttpPut] i [HttpDelete] per definir els mètodes HTTP que accepten.

El controlador de la nostra aplicació disposarà de dos mètodes públics. El primer és un mètode GET, utilitzat per obtenir un registre de la base de dades per identificador.

```
HttpGet("{itemNumber}")]
public ActionResult<LegoSet> GetLegoSet(string itemNumber)
{
    var legoSet = context.Sets.Find(itemNumber);
```

```
    if (legoSet == null)
    {
        return NotFound();
    }

    return legoSet;
}
```

Aquest mètode pot retornar o bé una acció o bé un tipus específic i és necessari proporcionar un identificador. En el mètode, es comença utilitzant la funció *Find()*, que realitza una consulta a la base de dades d'una entitat amb els valors de la clau principal especificats i aquesta entitat, si es troba, s'adjunta al context i es retorna. En el cas que no es trobi, retorna nul. Per tant, es comprova si és nul·la i, en cas afirmatiu, es retorna una resposta HTTP de *Not Found* amb codi d'estat 404 o, en cas contrari, és retorna el tipus especificat.

El segon mètode és de tipus POST, i s'utilitza per afegir un nou registre a la base de dades o per actualitzar-ne un d'existent.

```
[HttpPost]
public ActionResult<LegoSet> PostLegoSet(LegoSet legoSet)
{
    var existingLegoSet = context.Sets.Find(legoSet.ItemNumber);

    if (existingLegoSet == null)
    {
        return AddNewLegoSet(legoSet);
    }

    if (!LegoSetsAreEqual(legoSet, existingLegoSet))
    {
        return UpdateExistingLegoSet(legoSet, existingLegoSet);
    }
```

```
    return Conflict("Lego set already exists and has not been updated.");
}
```

Aquest mètode retorna una acció o un objecte del tipus especificat i és necessari que rebi un objecte del model. Es segueix el mateix procediment que en la funció anterior per comprovar si l'objecte rebut existeix a la base de dades o no.

En el cas que no existeixi, és a dir que sigui nul, es cridarà i retornarà la funció per afegir el registre a la base de dades, i en el cas que ja hi hagi sigut afegit anteriorment, es farà una comprovació per saber si s'ha modificat i, si s'ha modificat, es cridarà i retornarà la funció per actualitzar el registre o, si no s'ha modificat, s'ignorarà i es retornarà una resposta conflicte (HTTP 409 *Conflict*), indicant que l'objecte ja existeix a la base de dades i que no ha estat modificat.

El mètode públic anterior fa ús de tres mètodes privats. Un per afegir un nou registre, un per actualitzar un registre existent i l'últim per comprovar si dos registres tenen el mateixos valors en les propietats.

```
private ActionResult<LegoSet> AddNewLegoSet(LegoSet legoSet)
{
    try
    {
        context.Sets.Add(legoSet);
        context.SaveChanges();
        return CreatedAtAction("GetLegoSet", new { itemNumber = legoSet.ItemNumber }, legoSet);
    }
    catch (DbUpdateException ex)
    {
        return StatusCode(500, ex.Message);
    }
}
```

Aquesta funció privada retorna una acció o un objecte del tipus especificat, és necessari que rebi un objecte del model com a paràmetre i es crida en cas que l'objecte proporcionat no existeixi en la base de dades.

La funció utilitza un bloc *try-catch* per gestionar possibles excepcions que puguin sorgir durant l'execució del codi. Dins d'aquest bloc s'afegeix un nou registre a la col·lecció del context, es guarden els canvis a la base de dades i es retorna una resposta HTTP 201 *Created* amb el registre afegit utilitzant la funció GET anterior.

Si es produeix una excepció del tipus *DbUpdateException* durant l'execució del bloc *try*, el bloc *catch* captura l'excepció i retorna una resposta HTTP 500 *Internal Server Error*.

```
private ActionResult<LegoSet> UpdateExistingLegoSet(LegoSet legoSet, LegoSet existingLegoSet)
{
    try
    {
        context.Entry(existingLegoSet).CurrentValues.SetValues(legoSet);
        context.SaveChanges();
        return CreatedAtAction("GetLegoSet", new { itemNumber = legoSet.ItemNumber }, legoSet);
    }
    catch (DbUpdateException ex)
    {
        return StatusCode(500, ex.Message);
    }
}
```

Aquesta funció privada retorna una acció o un objecte del tipus especificat i és necessari que rebi com a paràmetres dos objectes del tipus especificat que fan referència a les noves dades i al conjunt existent que s'ha de modificar.

Dins del bloc *try*, s'obté el registre de dades corresponent a l'objecte existent, s'actualitzen els valors actuals amb els del nou objecte, es guarden els canvis a la base de dades i es retorna una resposta HTTP 201 *Created* amb el registre afegit utilitzant la funció GET anterior. Si es produeix una excepció del tipus *DbUpdateException* durant l'execució d'aquest bloc, el bloc *catch* captura l'excepció i retorna una resposta HTTP 500 *Internal Server Error*.

```
private static bool LegoSetsAreEqual(LegoSet legoSet, LegoSet oldLegoSet)
{
    PropertyInfo[] properties = typeof(LegoSet).GetProperties();
```

```
    foreach (PropertyInfo property in properties)
    {
        if (property.GetValue(legoSet)?.Equals(property.GetValue(oldLegoSet)) != true)
        {
            return false;
        }
    }

    return true;
}
```

La última funció privada i estàtica retorna un valor booleà i pren com a paràmetres dos objectes del tipus especificat i s'utilitza per comprovar si dos objectes tenen els mateixos valors.

Primer, s'obté un llistat de totes les propietats de la classe model fent ús de la classe *Reflection*. Llavors, es recorren aquestes propietats comparant-ne els valors dels dos objectes i si alguna propietat no és igual es retornarà fals. Si el bucle finalitza sense trobar cap propietat amb valors diferents, la funció retorna *true*, indicant que els dos objectes són iguals.

6.5. Encapsulació dels components en contenidors Docker

L'encapsulació dels components s'ha dut a terme en un fitxer Docker Compose general on queden especificats els tres serveis de l'aplicació, els secrets i la xarxa interna. A part, per poder encapsular tant l'script com la API, no és suficient amb només el fitxer Compose, també s'ha hagut de crear imatges personalitzades pels dos components.

6.5.1. Imatges personalitzades

Pyhton

Aquest *Dockerfile* crea una imatge basada en Python i Alpine Linux. Amb la instrucció FROM es defineix la imatge base des de la qual es partirà per a construir la nova imatge

Docker. S'utilitza la versió 3.12.1 de Python juntament amb la distribució Alpine Linux 3.19, que és una imatge molt lleugera i adequada per a contenidors.

Després, amb la instrucció USER, s'estableix l'usuari *root* com l'usuari que executarà les següents instruccions del *Dockerfile*, ja que algunes operacions, com la instal·lació de paquets, requereixen permisos d'administrador.

La instrucció WORKDIR estableix el directori de treball a la ruta especificada volent dir que qualsevol instrucció posterior que utilitzi un camí relatiu s'executarà des d'aquest directori. I afegint la instrucció VOLUME, es declara el directori com un volum de Docker, permetent que les dades d'aquest directori persisteixin encara que s'elimini el contenidor.

El comandament RUN executa *ensurepip* per comprovar i assegurar que l'instal·lador de paquets de Pyhton *pip* estigui instal·lat i després s'utilitza aquest instal·lador per obtenir els paquets necessaris (*Requests* i *Beautiful Soup*) sense utilitzar la memòria cau i ignorant les advertències de *root*.

Finalment, amb la instrucció CMD, es defineix la comanda per defecte que s'executarà quan es llanci un contenidor a partir d'aquesta imatge. Aquesta comanda executa el script del directori assignat utilitzant Python en mode *unbuffered*, significant que la sortida de Python no estarà en memòria intermèdia permetent veure els registres en temps real i evitant problemes de buffering en el contenidor.

```
FROM python:3.12.1-alpine3.19


USER root


WORKDIR /python-app

VOLUME /python-app


RUN python3 -m ensurepip && \

    pip3 install --no-cache --upgrade --root-user-action=ignore pip
requests beautifulsoup4


CMD ["python3", "-u", "script.py"]
```

.NET

Aquest *Dockerfile* crea una imatge Docker que inclou totes les eines necessàries per compilar i executar aplicacions .NET. Amb la instrucció FROM s'especifica la imatge base a utilitzar, que és una imatge del SDK de .NET versió 8.0.100 sobre una distribució Alpine Linux 3.18 per a l'arquitectura ARM.

Amb la instrucció USER es defineix l'usuari sota el qual s'executaran les següents instruccions del *Dockerfile*.

ENV s'utilitza per establir una variable d'entorn dintre del contenidor. Aquesta variable serà utilitzada per especificar el nom del fitxer DLL a executar. I amb la instrucció EXPOSE, s'indica que el contenidor escoltarà en el port 80. És important denotar que això no fa que el port sigui accessible des de fora del contenidor, sinó que el servei dins del contenidor utilitzi aquest port.

Com s'havia comentat abans, WORKDIR estableix el directori de treball a la ruta especificada i VOLUME crea un punt de muntatge per a un volum en aquest directori.

La instrucció RUN executa una comanda en temps de construcció del contenidor actualitzant tots els paquets d'Alpine a les seves versions més recents i eliminant la memòria cau per reduir la mida de la imatge.

Finalment, amb la instrucció CMD, s'especifica la comanda per defecte que s'executarà quan s'inicií el contenidor. Aquesta, executa el *runtime* de .NET per carregar i executar el fitxer DLL especificat per la variable d'entorn DLL_NAME, i escoltarà en totes les interfícies (http://*).

```
FROM mcr.microsoft.com/dotnet/sdk:8.0.100-alpine3.18-arm64v8

USER root

ENV DLL_NAME=default
EXPOSE 80

WORKDIR /dotnet-app
VOLUME /dotnet-app
```

```
RUN apk update && \
    apk upgrade && \
    rm -rf /var/cache/apk/*

CMD dotnet /dotnet-app/${DLL_NAME}.dll --urls "http://*"
```

Una vegada el fitxer *Dockerfile* és completat, ja es pot construir la imatge corresponent al fitxer. Per a fer-ho s'utilitza el següent comandament de Docker.

```
docker build {ruta del Dockerfile} -t {etiqueta de la imatge}
```

Això, construirà la imatge de la ruta on estigui guardat el *Dockerfile* amb una etiqueta (nom) i quedarà guardada a les imatges creades que després es podran usar per a crear contenidors. Per verificar que s'ha creat correctament es poden llistar les imatges amb el comandament *docker images*.

6.5.2. Fitxer Docker Compose

Com ja s'ha comentat, el fitxer Docker Compose d'aquest projecte contindrà tres serveis diferents: *database*, *API* i *script*. Aquests, quedaran definits en un l'element d'alt nivell del fitxer (*services*).

Els serveis tindran alguns atributs en comú, com el de *restart*, que definirà la política que s'aplicarà a la plataforma si el contenidor termina per algun motiu. En el cas dels tres serveis, s'ha definit que es reinicïi el contenidor independentment del codi de sortida, però que es deixi de reiniciar quan el servei s'aturi o s'elimini amb la política *unless-stopped*.

Els tres serveis també compartiran una mateixa xarxa per a què es puguin **comunicar** entre ells de manera interna. Aquesta xarxa quedarà definida en un altre element d'alt nivell del fitxer (*networks*) de la següent manera:

```
networks:
  net:
    name: lego_net
    driver: bridge
    ipam:
      config:
        - subnet: 192.168.250.0/24
          gateway: 192.168.250.1
```

Per tant, dins de cada servei s'utilitzarà l'atribut *networks* amb el nom de la xarxa creada per assignar una adreça IP estàtica.

Finalment, els serveis també compartiran l'atribut *mem_limit*, que configura el límit en la mida de memòria que un contenidor pot assignar-se.

Database

Començant amb cada servei en concret, el servei de la base de dades tindrà una imatge base oficial de PostgreSQL versió 16.1 basada en la distribució Alpine, que farà que sigui més lleugera.

Es farà el mapeig del port de PostgreSQL per poder accedir a la base de dades des de la xarxa local amb l'eina d'administració de bases de dades per poder fer proves i veure gràficament l'estructura i els registres emmagatzemats.

Posteriorment, s'assignaran les variables d'entorn de l'aplicació. **POSTGRES_PASSWORD** és una variable d'entorn requerida per utilitzar la imatge de PostgreSQL. Aquesta variable d'entorn estableix la contrasenya de superusuari. El superusuari per defecte és definit per la variable d'entorn POSTGRES_USER. **POSTGRES_USER** és una variable d'entorn opcional que crearà l'usuari especificat amb privilegis de superusuari. Si no s'especifica, es farà servir l'usuari per defecte, *postgres*. I la variable **POSTGRES_DB**, que és opcional i es pot utilitzar per definir un nom diferent de la base de dades per defecte que es crea quan s'inicia la imatge per primera vegada. Si no s'especifica, s'utilitzarà el valor de POSTGRES_USER.

En el cas d'aquest servei, per millorar la seguretat i portabilitat, s'ha optat per utilitzar *Secrets* com a alternativa per passar informació sensible a través de variables d'entorn.

Afegint **_FILE** darrera de les variables llistades anteriorment, s'obliga al script d'inicialització del contenidor a carregar els valors de les variables a partir d'un fitxer.

Per utilitzar *Secrets*, s'ha de crear un altre element d'alt nivell fora del de *services* en el fitxer Compose definint les referències a les dades sensibles que seran habilitades al servei.

```
secrets:
  db:
    file: ./environment/db.txt
  user:
    file: ./environment/user.txt
  password:
    file: ./environment/password.txt
```

Una vegada definits els secrets, s'actualitza el servei afegint l'atribut *secrets* per referenciar els secrets que el servei necessita. I a cada variable d'entorn se li assigna el valor */run/secrets/nom_del_secret* en comptes de mostrar la dada sensible.

Finalment, amb l'atribut *volumes*, primer es mapeja el directori de l'amfitrió on es vol que es guardi la informació de la base de dades al directori del contenidor, permetent la persistència de les dades entre reinicis del contenidor.

El segon volum, per altra banda, mapeja el fitxer SQL que conté el C*REATE* de la taula al directori del contenidor que s'executa automàticament quan es configura la base de dades per primera vegada, permetent així que la taula es creï sola en inicialitzar-se la base de dades.

```
CREATE TABLE IF NOT EXISTS "Sets" (
	"ItemNumber" VARCHAR NOT NULL,
	"Name" VARCHAR NOT NULL,
	"Parts" INT NOT NULL,
	"Theme" VARCHAR NOT NULL,
	"YearReleased" INT NOT NULL,

	PRIMARY KEY("ItemNumber")
);
```

El servei *database* sencer:

```
database:
    container_name: 'lego_database'
    image: postgres:16.1-alpine3.19
    restart: unless-stopped
    networks:
      net:
        ipv4_address: 192.168.250.2
    ports:
      - '54320:5432'
    environment:
      TZ: 'Europe/Madrid'
      POSTGRES_DB_FILE: /run/secrets/db
      POSTGRES_USER_FILE: /run/secrets/user
      POSTGRES_PASSWORD_FILE: /run/secrets/password
    secrets:
      - db
      - user
      - password
    mem_limit: 256MB
    volumes:
      - '/server_data/docker_volumes/web_scraping/lego/database:/var/lib/postgresql/data'
      - './init.sql:/docker-entrypoint-initdb.d/init.sql'
```

API

El següent servei especificat al fitxer Docker Compose és la API, la qual obtindrà la imatge personalitzada creada anteriorment.

Amb l'atribut *depends_on*, es pot expressar una dependència d'inici o apagat a un altre contenidor. En aquest cas, es vol que el contenidor del servei *API* s'iniciï i s'apagui una vegada el contenidor del servei *database* estigui iniciat o apagat.

També es mapejarà el port de la API per poder fer proves des de la xarxa local i el directori on es guardarà la informació de la API al directori que s'ha especificat a la creació de la imatge.

Com a variable d'entorn, s'assigna el valor a la variable que requereix el contenidor especificada a la imatge i que serà amb la que l'aplicació serà executada.

```
API:
    container_name: 'lego_api'
    image: pauserra/dotnet_http:8.0.100
    restart: unless-stopped
    depends_on:
      - database
    networks:
      net:
        ipv4_address: 192.168.250.3
    ports:
      - '8080:80'
    environment:
      TZ: 'Europe/Madrid'
      DLL_NAME: LegoAPI
    mem_limit: 512MB
    volumes:
      - '/server_data/docker_volumes/web_scraping/lego/API:/dotnet-app'
```

Script

L'últim servei del fitxer és el script de Pyhton. Aquest es basarà en la imatge personalitzada creada anteriorment i dependrà de la API.

Se li mapejarà el directori on es guardarà el script al directori que s'ha especificat a la creació de la imatge.

```
script:
    container_name: 'lego_script'
    image: pauserra/python:3.12.1
    restart: unless-stopped
    depends_on:
      - API
    networks:
      net:
        ipv4_address: 192.168.250.4
    environment:
      TZ: 'Europe/Madrid'
    mem_limit: 256MB
    volumes:
      - '/server_data/docker_volumes/web_scraping/lego/script:/python-app'
```

Una vegada completat el fitxer Docker Compose, per a construir els contenidors dels serveis que s'han especificat, es necessita utilitzar el següent comandament:

```
docker compose up
```

Si es vol executar el comandament de forma deslligada per poder utilitzar la consola sense haver d'aturar els contenidors es pot afegir l'etiqueta *-d* al final. Per altra banda, per comprovar que els contenidors s'han creat correctament els podem llistar utilitzant el comandament *docker ps*. I finalment, per aturar els contenidors s'utilitza *docker compose down*.

6.6. Esquemes del sistema

En el següent esquema del projecte es mostra visualment com s'han estructurat els components del sistema dintre de la Raspberry Pi 5.

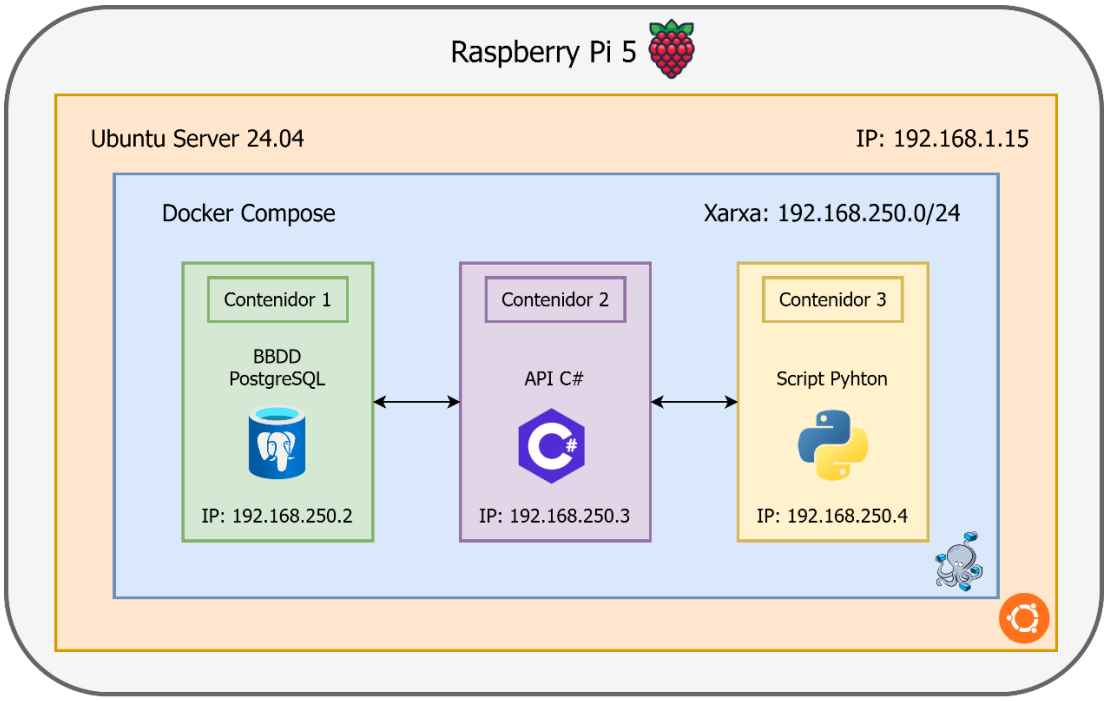

Figura 7. Esquema intern del sistema

Començant de l'interior del sistema cap enfora, s'ha creat els tres contenidors amb els seus components respectius de manera que estiguin a la mateixa xarxa interna declarada a la configuració del Docker Compose perquè es puguin comunicar entre ells sense necessitat d'exposar directament els serveis fora de l'entorn de Docker *(els serveis s'exposaran igualment amb el mapeig de ports amb la finalitat de depuració i demostració)*. Aquesta xarxa interna serà la 192.168.250.0/24 i cada un dels serveis agafarà una adreça IP assignada estàticament de la mateixa.

El contenidor amb el servei de l'script de Python serà el que es connecta a l'exterior per obtenir les dades de Internet i fer les peticions a la HTTP a la API. L'API actua com a intermediari i ofereix un punt d'entrada per l'script per interactuar amb la base de dades de manera segura i estructurada. Per tant, l'API envia consultes SQL a la base de dades per obtenir o actualitzar dades segons les sol·licituds que rep de l'script.

El sistema operatiu Ubuntu Server 24.04 base instal·lat a la Raspberry Pi 5 té assignada l'adreça IP 192.168.1.15 de la xarxa local (LAN) que s'utilitzarà per sortir a Internet a través de l'enrutador.

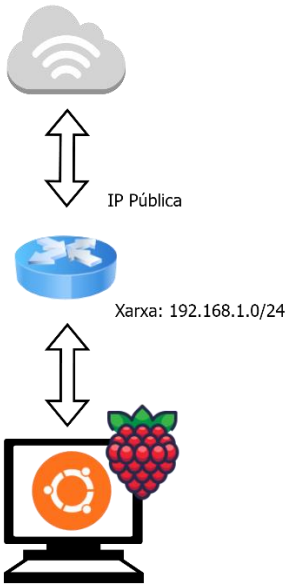

Figura 8. Esquema de la xarxa

7. Proves i anàlisi dels resultats

Per a fer les proves del sistema utilitzarem una pàgina web que conté informació sobre els diferents sets de LEGO Star Wars de l'any 2024 *(a l'script l'any queda establert de manera dinàmica fent ús de la llibreria DateTime per tant anirà canviant automàticament)*. La pàgina objectiu en qüestió és la següent:

https://rebrickable.com/sets/star-wars/2024/?sort_sets_by=0&sort_sets_dir=D&sets_page_size=200

Si accedim a la pàgina web i analitzem la seva estructura, es pot veure que disposa de una barra de cerca amb un sistema de filtratge i a sota, una quadrícula que conté una sèrie de contenidors amb les dades que es pretén extreure. Cada contenidor serà un registre que s'emmagatzemarà a la base de dades.

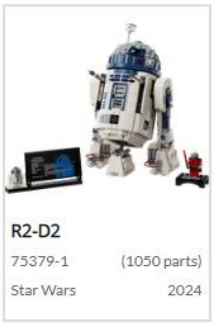

Figura 9. Exemple de registre a obtenir de la pàgina web

Per a començar amb el procés d'extracció utilitzant Beautiful Soup, com ja s'ha comentat a l'apartat del Desenvolupament del script, per extreure un conjunt d'elements de la pàgina web s'utilitza la funció *find_all* especificant el tipus d'element i la seva classe per filtrar els rellevants. En el cas d'aquesta pàgina web, si n'inspeccionem l'estructura, el tipus d'element que interessa és un *div* i la seva classe és *col-xs-6 col-sm-4 col-md-3 col-lg-2 js-set col-condensed*.

```
data = html_soup.find_all('div',
       class_='col-xs-6 col-sm-4 col-md-3 col-lg-2 js-set col-condensed')
```

Una vegada especificat el tipus d'element que es vol aconseguir, es dona per acabada la configuració i preparació de l'script d'aquesta pàgina en concret, ja que la resta del procés queda exposat a l'apartat de la Implementació del sistema.

Per a realitzar proves de l'eficiència del script en extreure les dades, s'ha utilitzat la llibreria *time* de Pyhton. Amb ella, es pot calcular el temps que triga en executar-se el tros de codi del script que extreu les dades de la pàgina web. Afegint les següents línies de codi (*start* a l'inici del codi que es vol calcular i *end* al final), es poden determinar i mostrar amb precisió els segons transcorreguts entre aquestes dues marques de temps.

```
start = time.time()
# Fragment de codi a mesurar
end = time.time()
print(end - start)
```

Havent executat el script una sèrie de vegades, s'han obtingut els següents resultats de les execucions en segons:

Execució	Temps (s)
1	0.961
2	0.922
3	1.043
4	0.905
5	0.924
6	1.055

Taula 4. Resultats execucions script

Analitzant els resultats, es pot determinar que l'script de Python triga al voltant de 1 segon en carregar la pàgina web, obtenir els elements amb les dades rellevants i en donar el format adequat a aquestes per ser enviades a la API.

Tot i això, s'han de tenir en compte dues variables. La primera és l'estat de la connexió a Internet, ja que carregar i obtenir la pàgina web amb la petició GET és el que més temps consumeix de tota l'execució, i diferències en l'estabilitat de la connexió podria fer variar bastant el temps d'execució.

L'altre variable és la quantitat de dades que s'han de processar, és a dir, quants més elements rellevants s'hagi d'obtenir més trigarà a realitzar-se l'execució i viceversa. Tanmateix, aquesta variable no té el pes que té la primera variable ja que el processament i manipulació de dades de Python una vegada obtingudes és tant ràpid que no hi ha una diferència rellevant entre processar-ne 1 o 100. I a més, en el cas d'aquest projecte no

s'han de processar grans quantitats de dades, per tant, es podria dir que tot el pes del temps d'execució cau en la càrrega i obtenció de la pàgina web.

D'altra banda, també s'han fet proves per conèixer l'eficiència del processament de les dades per part de la API. Utilitzant Visual Studio 2022 per depurar i establir punts d'interrupció i Postman per enviar peticions a la API manualment, s'han obtingut els següents resultats en mil·lisegons:

Execució	Temps (ms)
1	1110
2	22
3	19
4	14
5	14
6	12
7	13

Taula 5. Resultats execucions API

Tenint en compte que l'API ja està llesta per acceptar peticions i executant-se, aquests temps inclouen la connexió a la base de dades, el processament de les dades i la inserció d'aquestes a la base de dades.

Després de realitzar diverses proves, s'ha observat un patró en el temps d'execució de les peticions a l'API. La primera execució triga aproximadament 1 segon, mentre que les execucions posteriors es completen en tan sols 15 o 20 mil·lisegons. Aquesta diferència s'explica pel fet que, durant la primera connexió amb la base de dades, l'API necessita establir i configurar la connexió, un procés que requereix un temps addicional. Un cop la connexió s'ha establert, es manté activa, i per tant, les següents peticions poden processar-se més ràpidament. Això explica la disparitat de temps entre la primera execució i les següents.

Per acabar, també s'ha mesurat el temps que necessiten els contenidors Docker dels components per desplegar-se i per aturar-se.

```
root@ubuntu-server:~/docker/web_scraping/lego# docker compose up -d
[+] Running 4/4
 ✓ Network lego_net          Created                              1.6s
 ✓ Container lego_database   Started                              0.9s
 ✓ Container lego_api        Started                              0.9s
 ✓ Container lego_script     Started                              1.1s
```

```
root@ubuntu-server:~/docker/web_scraping/lego# docker compose down
[+] Running 4/4
 ✓ Container lego_script     Removed                             12.6s
 ✓ Container lego_api        Removed                             12.4s
 ✓ Container lego_database   Removed                             12.4s
 ✓ Network lego_net          Removed                              0.2s
```

En resum, el temps total necessari per arrencar el sistema per primera vegada és aproximadament de **6 segons**. Aquest període inclou diverses tasques com la creació dels contenidors per a cada component, l'establiment de la xarxa interna per a la comunicació entre ells, la obtenció de les dades de la pàgina web, la configuració de la connexió entre l'API i la base de dades, i finalment, la inserció i emmagatzematge d'aquestes dades.

És important subratllar que aquest temps de 6 segons només s'aplica a la primera arrencada. Posteriorment, el sistema està programat per executar-se automàticament a intervals regulars. Com a resultat, les extraccions i emmagatzematges de dades que es realitzin després de la primera execució trigaran només aproximadament **1 segon** cadascuna.

8. Conclusions

8.1. Assoliment dels objectius

En aquest apartat es presenten les conclusions generals del projecte, s'avalua l'assoliment dels objectius inicials i s'analitza críticament la planificació i la metodologia seguides.

Conclusions del treball

El projecte ha estat una experiència enriquidora que ha proporcionat una comprensió profunda dels processos implicats en la creació d'un sistema automatitzat d'extracció de dades web. Algunes de les principals lliçons apreses inclouen:

- La integració de tecnologies diverses: l'ús de múltiples tecnologies com Python per al web scraping, PostgreSQL per a l'emmagatzematge de dades, i C# per a la creació de l'API han demostrat la importància de la interoperabilitat i la integració efectiva entre diferents eines i llenguatges de programació.

- La importància de la documentació: mantenir una documentació adequada i detallada durant tot el procés de desenvolupament ha estat crucial per facilitar la comprensió i el manteniment del sistema.

- L'eficiència de la contenidorització: la contenidorització amb Docker ha simplificat enormement el desplegament i la gestió del sistema, destacant el valor d'aquesta tecnologia en projectes de desenvolupament.

Reflexió sobre l'assoliment dels objectius

En general, el projecte ha assolit amb èxit els objectius plantejats inicialment. Els objectius específics establerts eren:

- Instal·lar i configurar un Ubuntu Server en una Raspberry Pi 5: L'objectiu s'ha assolit amb èxit, proporcionant una plataforma robusta per al sistema.

- Crear una base de dades PostgreSQL per emmagatzemar les dades recopilades: La base de dades ha estat creada i s'utilitza per a emmagatzemar de manera organitzada les dades extretes.

- Desenvolupar una API REST amb C#: S'ha desenvolupat una API funcional que permet l'accés i manipulació de les dades emmagatzemades.

- Desenvolupar un script amb Python i Beautiful Soup per extreure les dades de la pàgina web: L'script s'ha creat i funciona segons les especificacions.

- Encapsular cadascun dels components en contenidors Docker: Els components han estat encapsulats amb èxit per assegurar la portabilitat i la facilitat de desplegament del sistema.

- Implementar un sistema automatitzat d'extracció de dades: S'ha completat de manera exitosa, amb un sistema que pot extreure dades de manera eficient i precisa.

- Documentar adequadament tots els components i processos: La documentació ha estat exhaustiva, detallant des de la instal·lació i configuració fins al desenvolupament i desplegament dels components.

Tanmateix, hi ha hagut alguns desafiaments durant el procés que han impactat parcialment l'assoliment complet de certs objectius menors. Per exemple, algunes funcionalitats avançades de l'API han requerit més temps del previst, i encara es podrien fer millores en la gestió d'errors. Això reflecteix la complexitat del treball i les oportunitats de millora contínua.

Anàlisi crític del seguiment de la planificació i la metodologia

La planificació inicial va establir un calendari detallat que incloïa diverses etapes del projecte. Tot i que en general s'ha seguit la planificació, hi ha hagut algunes desviacions:

- Adequació de la metodologia: la metodologia prevista ha estat adequada per a la majoria de les tasques. La divisió del projecte en fases clares i l'ús de tecnologies de codi obert han facilitat l'organització i l'execució del treball.

- Adaptacions: durant el desenvolupament, va ser necessari ajustar el cronograma original per acomodar temps addicional per a les proves i la depuració del sistema.

- Flexibilitat en la planificació: la flexibilitat en la planificació ha estat clau per abordar els imprevistos. Per exemple, la complexitat tècnica de certs aspectes del

desenvolupament de l'API va requerir un replantejament i una revisió addicional que no s'havien previst inicialment.

En conclusió, malgrat alguns ajustaments necessaris en el procés de desenvolupament, el projecte ha aconseguit complir amb els seus objectius principals i ha proporcionat una base sòlida per a futures ampliacions i millores.

8.2. Possibles extensions o millores futures

Tot i que el sistema implementat és funcional en l'extracció de dades web i compleix amb els objectius que s'havien plantejat a l'inici del projecte, durant el desenvolupament d'aquest s'han anat plantejant potencials extensions per ampliar la seva funcionalitat.

La primera d'aquestes millores podria ser la creació d'un **registre d'errors i d'activitat** del sistema. Actualment, s'ha de fer ús de comandaments de Docker per comprovar que el sistema està en correcte funcionament o si hi ha hagut alguna falla en extreure les dades o en emmagatzemar-les, per exemple. Per tant, es podria incloure una nova funcionalitat que gestionés i emmagatzemés de forma amigable una sèrie de registres útils per la identificació de problemes o per assegurar-se que el sistema està extraient dades satisfactòriament.

Per altra banda, també s'ha plantejat la possibilitat d'implementar un **sistema de notificacions automàtiques** cada vegada que s'aconsegueixin noves dades, ja que actualment l'única manera de saber si s'ha afegit un nou registre és accedint a la base de dades i comprovar-ho manualment. És per això que es podria fer ús de plataformes com Telegram, que faciliten la creació de Bots altament personalitzables que es poden afegir a un xat privat i es poden configurar per enviar un missatge quan es rebi nova informació extreta de la web.

Aquestes propostes són exemples d'extensions que es podrien implementar per millorar el manteniment i l'experiència del projecte final. Tot i això, no s'han pogut dur a terme ja que no es van incloure en la temporització inicial del projecte i no hi ha hagut temps per desenvolupar-les.

9. Glossari

Script: conjunt d'instruccions escrites en un llenguatge de programació que es poden executar automàticament per tal de simplificar tasques en un ordinador o en una aplicació.

API REST (*Representational State Transfer*): tipus d'API que permet la comunicació entre diferents sistemes de manera estandarditzada i utilitza el protocol HTTP per a la transmissió de dades.

SBC (*Single Board Computer*): ordinadors que tan sols requereixen d'una placa computadora amb un circuit per funcionar.

CLI (*Command-Line Interface*): la interfície de línia d'ordres és un mètode per manipular amb instruccions escrites un programa. S'interacciona sense gràfics, només amb el text cru.

Compilar: procés de convertir o traduir el codi font, que és escrit en un llenguatge de programació de més alt nivell, en un codi objecte o binari que la màquina pot executar directament. Aquest procés és dut a terme per un programa especial anomenat compilador.

Multiplataforma: programari, ja sigui un sistema operatiu, llenguatge de programació programa, etc. que pot ser executat en diverses plataformes.

Repositori: espai d'emmagatzematge centralitzat on es guarda tot el codi font, així com altres recursos rellevants per a un projecte.

Versió LTS (*Long Term Support*): versions de programari dissenyats per a tenir suport durant un període més llarg que el normal, possibilitant als usuaris treballar en projectes a llarg termini amb programari que confereix estabilitat i minimitza els errors.

EOL (*End Of Life*): final del cicle de vida del producte que impedeix que els usuaris rebin actualitzacions, que indiquen que el producte està al final de la seva vida útil (des del punt de vista del venedor).

IDE (*Integrated Development Environment*): entorn de desenvolupament que agrupa diferents funcions en un sol programa (editor de codi, compilador, depurador i un programa de disseny d'interfície gràfica).

Endpoint: punt específic al qual un client pot accedir per interactuar amb els serveis que proporciona l'API. Un *endpoint* és l'adreça URL específica que una API proporciona per realitzar una acció concreta.

Mètodes HTTP: operacions que es poden realitzar en recursos accessibles a través d'una API. Cada mètode defineix una acció específica que es pot aplicar a un recurs, com obtenir, actualitzar, crear o eliminar dades (GET, POST, PUT, DELETE, entre d'altres).

CRUD: acrònim que descriu les quatre operacions bàsiques que es poden realitzar sobre dades en el context de les bases de dades i les aplicacions web (Create, Read, Update, Delete). Fa referència als mètodes HTTP anteriors.

Alpine: és una distribució de Linux molt lleugera i eficient que permet crear contenidors en un entorn Linux amb una àmplia selecció de paquets del repositori requerint un espai mínim d'emmagatzematge. Utilitzant aquesta distribució s'estalvia espai i recursos del sistema.

Excepció: condició anòmala o excepcional que requereix un processament especial durant l'execució d'un programa. Una excepció trenca el flux normal d'execució.

Punt d'interrupció: lloc d'aturada o pausa intencionada en un programa, establert amb finalitats de depuració.

10. Bibliografia i webgrafia

- Linux y GNU - Proyecto GNU - Free Software Foundation
  https://www.gnu.org/gnu/linux-and-gnu.es.html

- ¿Qué es GNU/Linux?
  https://www.debian.org/releases/stable/s390x/ch01s02.es.html

- Linux distribution - Wikipedia
  https://en.wikipedia.org/wiki/Linux_distribution

- GNU/Linux - Wikipedia
  https://es.wikipedia.org/wiki/GNU/Linux

- Web scraping - Wikipedia
  https://en.wikipedia.org/wiki/Web_scraping

- ¿Qué Es el Web Scraping? Cómo Extraer Legalmente el Contenido de la Web
  https://kinsta.com/es/base-de-conocimiento/que-es-web-scraping/

- Raspberry Pi Documentation
  https://www.raspberrypi.com/products/raspberry-pi-5/

- Ubuntu - Viquipèdia
  https://ca.wikipedia.org/wiki/Ubuntu#Ubuntu_Server

- Ubuntu for ARM
  https://ubuntu.com/download/server/arm

- PostgreSQL: Documentation
  https://www.postgresql.org/docs/current/

- PostgreSQL - Viquipèdia
  https://ca.wikipedia.org/wiki/PostgreSQL

- About | Alpine Linux
  https://alpinelinux.org/about/

- Información general - A tour of C# | Microsoft Learn
  https://learn.microsoft.com/ca-es/dotnet/csharp/tour-of-csharp/overview

- Introducción a .NET - .NET | Microsoft Learn
  https://learn.microsoft.com/ca-es/dotnet/core/introduction?WT.mc_id=dotnet-35129-website

- Docker - Viquipèdia
  https://ca.wikipedia.org/wiki/Docker

- Docker Compose overview | Docker Docs
  https://docs.docker.com/compose/

- Overview of the get started guide | Docker Docs
  https://docs.docker.com/guides/get-started/

- Documentación de ASP.NET | Microsoft Learn
  https://learn.microsoft.com/es-es/aspnet/core/?view=aspnetcore-8.0

- Documentación de la familia de productos de Visual Studio | Microsoft Learn
  https://learn.microsoft.com/es-es/visualstudio/?view=vs-2022

- Imagen de Docker y contenedor: diferencia entre tecnologías de implementación de aplicaciones - AWS
  https://aws.amazon.com/es/compare/the-difference-between-docker-images-and-containers/

- Dockerfile reference | Docker Docs
  https://docs.docker.com/reference/dockerfile/

- compose-spec/spec.md at master · compose-spec/compose-spec · GitHub
  https://github.com/compose-spec/compose-spec/blob/master/spec.md

- postgres – Official Image | Docker Hub
  https://hub.docker.com/_/postgres

- DBeaver Community | Free Universal Database Tool
  https://dbeaver.io/

- Bots: An introduction for developers
  https://core.telegram.org/bots

11. Annexos

11.1. Script en Pyhton

```
from bs4 import BeautifulSoup
import requests
import datetime
import re
import time

SLEEP_MINUTES = 10

def main():
    while True:
        get_response = requests.get(f'https://rebrickable.com/sets/star-
wars/{datetime.date.today().year}/?sort_sets_by=0&sort_sets_dir=D&sets_page_size=200')

        if get_response.status_code == 200:
            html_soup = BeautifulSoup(get_response.text, 'html.parser')
            data = html_soup.find_all('div', class_='col-xs-6 col-sm-4 col-md-3 col-
lg-2 js-set col-condensed')

            for div in data:
                lines = []

                for line in div.text.splitlines():
                    if line.strip():
                        lines.append(line)

                legoSet = dict(
                    itemNumber = lines[1],
                    name = lines[0],
                    parts = int(re.search(r'\d+', lines[2]).group()),
                    theme = lines[3],
                    yearReleased = int(lines[4]),
                )

                requests.post(url="http://192.168.250.3/api/LegoSets", json=legoSet)
        else:
            print("Failed to retrieve HTML: ", get_response.status_code)

        time.sleep(SLEEP_MINUTES * 60)

main()
```

11.2. API en C#

```
using LegoAPI.Context;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

builder.Configuration.AddJsonFile("./appsettings.json");

if (builder.Environment.IsProduction())
{
    builder.Services.AddDbContext<AppDbContext>(options =>
options.UseNpgsql(builder.Configuration.GetConnectionString("lego")));
}
else if (builder.Environment.IsDevelopment())
{
    builder.Services.AddDbContext<AppDbContext>(options =>
options.UseNpgsql(builder.Configuration.GetConnectionString("lego_dev")));
}

builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();

var app = builder.Build();

app.UseAuthorization();
app.MapControllers();
app.Run();
```

Program.cs

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning",
      "Microsoft.EntityFrameworkCore.Database.Command": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "lego": "User Id=pau;Password=;Server=192.168.250.2;Port=5432;Database=lego;",
    "lego_dev": "User Id=pau;Password=;Server=192.168.1.15;Port=54320;Database=lego;"
  }
}
```

appsettings.json

```
using LegoAPI.Models;
using Microsoft.EntityFrameworkCore;

namespace LegoAPI.Context
{
    public class AppDbContext(DbContextOptions<AppDbContext> options) : DbContext(options)
    {
        public DbSet<LegoSet> Sets { get; set; }
    }
}
```

AppDbContext.cs

```
using System.ComponentModel.DataAnnotations;

namespace LegoAPI.Models
{
    public class LegoSet
    {
        [Key]
        public required string ItemNumber { get; set; }

        [Required]
        public required string Name { get; set; }

        [Required]
        public required int Parts { get; set; }

        [Required]
        public required string Theme { get; set; }

        [Required]
        public required int YearReleased { get; set; }
    }
}
```

LegoSet.cs

```
using LegoAPI.Context;
using LegoAPI.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Reflection;

namespace LegoAPI.Controllers
{
```

```
    [ApiController]
    [Route("api/[controller]")]
    public class LegoSetsController(AppDbContext context) : ControllerBase
    {
        // GET: api/LegoSets/75858-1
        [HttpGet("{itemNumber}")]
        public ActionResult<LegoSet> GetLegoSet(string itemNumber)
        {
            var legoSet = context.Sets.Find(itemNumber);

            if (legoSet == null)
            {
                return NotFound();
            }

            return legoSet;
        }

        // POST: api/LegoSets
        [HttpPost]
        public ActionResult<LegoSet> PostLegoSet(LegoSet legoSet)
        {
            var existingLegoSet = context.Sets.Find(legoSet.ItemNumber);

            if (existingLegoSet == null)
            {
                return AddNewLegoSet(legoSet);
            }

            if (!LegoSetsAreEqual(legoSet, existingLegoSet))
            {
                return UpdateExistingLegoSet(legoSet, existingLegoSet);
            }

            return Conflict("Lego set already exists and has not been updated.");
        }

        private ActionResult<LegoSet> AddNewLegoSet(LegoSet legoSet)
        {
            try
            {
                context.Sets.Add(legoSet);
                context.SaveChanges();
                return CreatedAtAction("GetLegoSet", new { itemNumber =
legoSet.ItemNumber }, legoSet);
            }
            catch (DbUpdateException ex)
            {
                return StatusCode(500, ex.Message);
            }
        }
```

```
        private ActionResult<LegoSet> UpdateExistingLegoSet(LegoSet legoSet, LegoSet
existingLegoSet)
        {
            try
            {
                context.Entry(existingLegoSet).CurrentValues.SetValues(legoSet);
                context.SaveChanges();
                return CreatedAtAction("GetLegoSet", new { itemNumber =
legoSet.ItemNumber }, legoSet);
            }
            catch (DbUpdateException ex)
            {
                return StatusCode(500, ex.Message);
            }
        }

        private static bool LegoSetsAreEqual(LegoSet legoSet, LegoSet existingLegoSet)
        {
            PropertyInfo[] properties = typeof(LegoSet).GetProperties();

            foreach (PropertyInfo property in properties)
            {
                if
(property.GetValue(legoSet)?.Equals(property.GetValue(existingLegoSet)) != true)
                {
                    return false;
                }
            }

            return true;
        }
    }
}
```

LegoSetsController.cs