UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Escola Politècnica Superior d'Enginyeria de Manresa

Millora d'un sistema de recol · lecció de dades

Migració de la plataforma a Raspberry Pi i millora de l'API

Manresa, 2 de maig de 2023

treball de fi de grau que presenta

Francesc Arrufí Fernández

en compliment dels requisits per assolir el

Grau en Enginyeria de Sistemes TIC

Direcció: Jordi Bonet Dalmau

«One day a tortoise will learn how to fly.»
Terry Pratchett
— Small Gods

Agraïments

Als meus pares, que m'han fet costat sempre de forma incondicional.

A Ingimec, que m'ha permès fer el meu treball de fi de grau dins la seva empresa i en tot moment s'ha mostrat comprensiva i atenta.

Als meus professors, i en especial al director d'aquest treball, Jordi Bonet Dalmau, per aguantar totes les preguntes que li he fet, inclús les més banals.

Als meus companys de carrera, en David Comabella, en Joan Marc Serrano i en Jordi Salguero, amb vosaltres he fet quasi totes les assignatures i pràctiques i si estic ara aquí és pel vostre suport. També a tots aquells estudiants que m'han ajudat en moments puntuals, com en Hafid o en Toni Sbert. Sense tots ells ara no seria aquí, i per això us dono els més sincers agraïments.

I en especial al meu tiet, el Dr. Manuel Fernández López, que m'ha donat tota la seva experiència d'una vida dedicada a la investigació i la docència. T'estic molt agraït.

Resum

Any 2019: Ingimec rep l'encàrrec d'un client de Polònia. S'ha de crear un sistema de visualització de dades per a una de les seves línies. Ingimec encarrega el projecte a una altra empresa. Neix el projecte 4246.

Any 2020: a l'empresa no agrada el resultat d'aquesta decisió. Decideix deixar el projecte al seu departament encarregat de la programació dels equips que controlen les línies de producció que dissenya. Aquest decideix continuar aplicant les mateixes tecnologies que l'empresa externa. Arriba l'any vint-i-u i el projecte s'acaba abandonant a causa de canvis en l'estructura del departament i l'ocasional falta de personal.

Any 2022: arriben noves incorporacions al departament. Es decideix enfocar el projecte des d'un altre prisma. Ara ja no és un encàrrec per a un client, és un esquer que ha de permetre captar-ne de nous. S'apliquen millores i canvis i s'implementa un prototip en un entorn industrial.

Any 2023: els costos derivats de les decisions tècniques anteriors fan inviable el projecte. Es decideix començar de zero i trencar amb tot el treball previ. Abans era Azure, ara és Flask. Anteriorment era una ESP32 amb entrades i sortides físiques i limitades, ara és una Raspberry Pi amb wifi que es comunica amb els PLC mitjançant Ethernet. Era C, .NET i Model-Vista-Controlador; ara és Python, HTML i JavaScript. A l'inici es basava en els productes de Microsoft, ara es basa en Linux i codi obert.

Hi ha molts projectes amb un objectiu semblant. Formes increïbles de fer-los. Maneres totalment oposades de crear una mateixa cosa. Aquest és un projecte de tants, fet d'una altra manera.

Aquest és el meu treball de fi de grau.

Abstract

Year 2019: Ingimec receives an order from a customer in Poland. A data display system must be created for one of their production lines. Ingimec entrusts the project to an external company, and the 4246 project is born.

Year 2020: Ingimec is not satisfied with the outcome of the external company's work, so they decide to assign the project to their in-house programming department responsible for designing and programming the production lines. This department decides to continue using the same technologies as the external company. However, due to changes in the department's structure and occasional lack of staff, the project is eventually abandoned.

Year 2022: With new additions to the company, they decide to approach the project from a different perspective. Instead of being solely an order for a customer, the project is seen as an opportunity to attract new customers. Improvements and changes are made, and a prototype is implemented in an industrial environment.

Year 2023: The costs associated with previous technical decisions make the project unfeasible. It is decided to start from scratch and break with all previous work. Previously based on Azure, now it is based on Flask. Previously using an ESP32 with limited physical inputs and outputs, now they use a Desbarra with Wi-Fi that communicates with PLCs via Ethernet. It was previously developed with C, .NET, and Model-View-Controller, and now it uses Python, HTML, and JavaScript. Initially based on Microsoft products, now it is based on Linux and open-source technologies.

There are many projects with a similar goal. Amazing ways to make them. Totally opposite ways of creating the same thing. This is a project of many, done in a different way.

This is my graduation thesis.

Índex

Índex de figures

Índex de taules

Part I.

Memòria

1. Introducció

Ingimec va ser contractada el 2019 per dotar un dels seus productes d'un sistema eficaç de visualització de dades, estil els quadres de comandament que s'acostumen a veure en les plantes de producció. Segons el client, aquest sistema havia d'estar basat en el núvol, mostrar les dades de les diverses màquines de la línia de producció en una interfície web amigable i adaptativa, calcular l'efectivitat total dels equips i guardar-ho tot en una base de dades. Naixia així el projecte 4246.

En aquell moment es va decidir subcontractar una empresa externa per al seu disseny i implementació, la qual va decidir basar-ho tot en la plataforma Microsoft Azure. Azure és una plataforma basada en el núvol creada per Microsoft i que ofereix, entre d'altres, sistemes de computació al núvol, gestió de dades, accés a servidors, emmagatzematge... La gestió de dades estaria basada en Cosmos DB, un servei també creat per Microsoft que ofereix bases de dades distribuïdes. Això no obstant, totes aquestes decisions van comportar, a la llarga, un sobrecost inesperat perquè entre molts altres factors Azure basa la seva tarifa en el nombre de transaccions que es fan, de forma que el cost s'anava incrementant com més dades s'enviaven al servidor.

Posteriorment, Ingimec es va adonar que aquesta no era la millor solució: dependre d'una empresa externa com Microsoft, amb un sistema que plantejava uns costos poc raonables per un projecte com aquest, obligava a encarir el preu final del producte que es volia vendre. Des d'un principi aquest projecte es va plantejar com un afegit a les màquines fabricades i dissenyades per l'empresa que tindria de captar futurs clients. Per fer-ho, el sistema ideat hauria de ser flexible, ampliable i barat, i aquests objectius no es complirien en un sistema com l'inicialment plantejat.

És per aquest motiu que ara s'ha decidit començar el projecte de nou, basant-ho tot en plataformes i sistemes lliures i preferiblement de codi obert. Per fer-ho, s'han plantejat diversos canvis que obliguen a dissenyar-ho tot des de zero. Ara, la base de dades Cosmos DB s'ha canviat per una base de dades SQLite3 que res té a veure amb l'anterior; les comunicacions entre els PLC que controlen les màquines i el servidor, inicialment plantejades per funcionar amb una esp32, ara s'han canviat per unes més simples programades amb Python; un servidor Flask i una aplicació web creats des de zero substitueixen el servidor d'Azure i els càlculs que es volien calcular i visualitzar s'han replantejat i modificat per fer-los més robustos de cara al futur. Una Raspberry PI ha de fer de nexe entre tots aquests punts.

Aquest treball recull tot aquest procés, el de migració d'una plataforma privativa i amb uns costos desproporcionats per un sistema creat des de zero per Ingimec, flexible i basat en codi obert. Des de la planificació de les tecnologies emprades i els càlculs realitzats, fins a la implementació final — que compren tant l'emmagatzematge de les dades, les comunicacions amb els PLC que controlen la maquinària dissenyada internament i l'execució dels càlculs plantejats com el disseny i implementació del servidor Flask i la interfície Web — es recullen en un document que mostra tot el treball fet i que poc té a veure amb uns objectius similars però implementats des d'un altre prisma totalment diferent.

2. Antecedents

L'origen d'aquest treball és el projecte 4246 d'Ingimec, el qual ha anat passant de mans i que mai s'ha acabat de definir. L'objectiu inicial d'aquest projecte és la creació d'un sistema de control visual accessible des de qualsevol entorn web, així com la conseqüent creació d'una base de dades i d'un sistema de control/configuració en temps real.

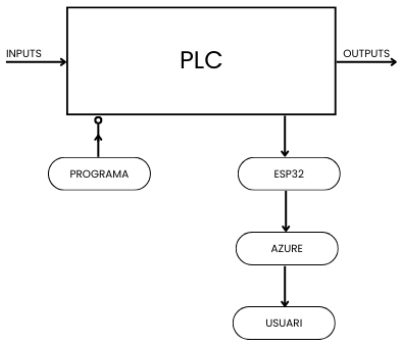

Figura 2.1.: En la primera implementació, es va fer servir una ESP32 per enviar les dades a un servidor basat en els serveis de Microsoft. L'ESP32 recollia les dades mitjançant les seves limitades entrades i sortides digitals i les enviava al servidor de Microsoft mitjançant wifi.

La primera implementació — Figura 2.1 — d'aquest projecte és la creació d'un sistema Azure coexistent a una base de dades Cosmos DB. Cosmos DB és una base de dades noSQL distribuïda lligada intrínsecament a la plataforma de computació des del núvol Azure, ambdós implementats per Microsoft. Això permet una ràpida configuració inicial, tot i que lligada als productes de Microsoft. El fet que aquestes plataformes no són lliures i que el seu cost ve determinat arbitràriament pels desitjos de Microsoft, han fet inviable continuar en aquesta direcció.

En Azure, el servei d'emmagatzematge de BLOB té un cost associat amb la transferència de dades, que es basa en la quantitat de dades transferides, així com en la ubicació geogràfica d'aquestes. A més, també hi ha costos associats amb la transferència de dades a través de les xarxes virtuals d'Azure, com en el tràfic sortint des d'una xarxa virtual a Internet [Mic23].

Amb aquest sistema, les dades s'envien a través d'un dispositiu basat en un ESP32 i desenvolupat per l'empresa bagenca Industrial Shields que envia les dades que recull a través de les seves entrades digitals cap al servidor Azure mitjançant una connexió wifi. A causa de la limitació física que té aquest dispositiu en comptar amb entrades digitals limitades, i als costos comentats anteriorment, s'ha optat per implementar una segona opció.

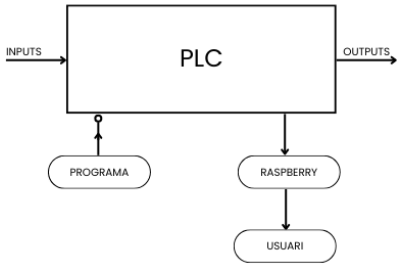

Figura 2.2.: La segona implementació, en canvi, treu els serveis de Microsoft i l'ESP32 i ho canvia directament per una Raspberry PI que fa totes dues funcions. Aquest dispositiu llegeix les dades del PLC mitjançant Ethernet, i disposa d'un segon port a més de wifi per si es vol activar la connexió a Internet amb un sistema o altre

La segona implementació — Figura 2.2 —, recollida en aquest treball, ha estat la creació del mateix sistema a través d'un servidor Linux allotjat en una Raspberry PI. S'ha optat per una Raspberry PI degut al seu poc cost i a que és un sistema que conté potència suficient per a complir l'objectiu proposat. En aquest punt, la connexió amb els PLC és a través d'Ethernet, el qual és més potent que les entrades digitals del dispositiu d'Industrial Shields. Això també permet una connexió directa amb els PLC, així com una major flexibilitat i escalabilitat de cara al futur. D'aquesta forma, si es vol millorar i afegir funcionalitats al sistema només s'ha d'incrementar el bus de bytes enviats des del PLC o canviar la Raspberry PI per un ordinador més potent.

Com es pot observar, el primer camí i el segon no són compatibles. Això comporta que s'hagi de tornar a fer tot de nou, començant des de la tramesa de dades del PLC cap a la Raspberry, passant per la creació de la base de dades i acabant amb el servidor Web allotjat en la mateixa Raspberry PI.

3. Càlculs, requeriments i competència

3.1. Càlculs

La finalitat d'aquest projecte és proporcionar un sistema de visualització atractiu i en temps real del rendiment d'una fàbrica, i per fer-ho és necessari calcular l'OEE. Aquest és un index percentual que mesura la productivitat de les fàbriques mitjançant tres paràmetres: disponibilitat, rendiment i qualitat. Descrit per primer cop per Seiichi Nakajima el 1982 en un article en japonès anomenat *TPM tenkai* i publicat per l'Institut Japonès de Manteniment de Plantes (JIPM)[MM13] i posteriorment editat en anglès [Nak89], és l'indicador més acceptat per mesurar l'eficiència i productivitat d'una planta.

3.1.1. Disponibilitat

Es calcula dividint el temps que la màquina ha estat produint entre el temps total. El temps total és la suma del temps produint més el temps de les avaries. El temps de les parades planificades és aquell que comprèn el temps perdut a causa d'aquelles parades que han estat programades amb anterioritat — per exemple l'hora d'esmorzar o els manteniments.

$$\text{Disponibilitat} = \frac{t_t}{t_T} \cdot 100$$

$$t_T = \text{temps treballat} - \text{temps planificat}$$

$$t_t = t_T - \text{temps averies}$$

Aquest paràmetre és el més difícil de calcular, degut principalment a què hi ha múltiples factors que poden afectar el temps que la fàbrica està treballant. Des d'un dispositiu com el que s'està dissenyant no és possible veure el que hi passa, ni tampoc si hi ha hagut algun problema inesperat que ha obligat a parar els equips. Per aquest motiu, en el DB del PLC s'ha afegit un bit *Manteniment* que permet precisar el càlcul del temps de parada planificada. També s'ha afegit un bit *Error* que permet el mateix per a les parades provocades per avaries.

En el càlcul es contempla que l'escriptura a la base de dades es fa a intervals constants i planificats, cada t segons. Es considera que les variacions que hi poden haver en aquest temps són menyspreables per al càlcul final, ja que el temps de cicle és aproximadament de 10 segons per peça.

Tenint en compte tot l'anterior, i que l'OEE es calcula per torns, el càlcul ha de quedar de la següent forma:

Temps total de treball t_w

Correspon al temps que s'ha estat treballant des de que el torn ha començat. És el resultat de restar a l'hora actual h l'hora en què ha començat el torn h_0.

$$t_w = h - h_0$$

Temps de parades planificades t_b

És el temps que previsiblement s'està sense produir. A la taula de torns de la base de dades s'ha afegit una columna *Break_time* que guarda el temps de descans total del torn. En el DB del PLC també s'ha configurat un bit *Manteniment* per tal de considerar en el càlcul aquest factor. D'aquesta forma, el càlcul del temps perdut per les parades que han estat planificades queda de la següent manera:

Primer es calcula el temps perdut a causa dels descansos t_r. Per fer-ho es multiplica b — *Break_time* — de la base de dades per t_w i es divideix pel temps total de durada del torn t_s.

$$t_r = b \cdot \frac{t_w}{t_s}$$

Després es calcula el temps perdut a causa dels manteniments. Per això s'ha configurat un bit *Manteniment*. Se suposa que l'operari de la fàbrica, abans de fer qualsevol acció, ha d'activar la corresponent opció des del PLC. Això posa el bit a «1». El programa de comunicació amb el PLC llegeix aquest valor i ho tracta amb conseqüència. D'aquesta forma el temps perdut per culpa dels manteniments t_m és el producte de totes les vegades que s'ha escrit aquest bit a la base de dades m pel temps que passa entre escriptura i escriptura t, que és constant.

$$t_m = m \cdot t$$

Finalment només queda sumar aquests dos resultats.

$$t_b = t_r + t_m$$

Temps d'avaries t_e

Si la producció ha sofert alguna avaria, mesura el temps que ha afectat la producció. Per calcular-ho s'ha configurat un bit *Error* que s'activa quan el PLC genera algun error. El programa de comunicació amb el PLC el llegeix i guarda a la base de dades (com en el bit *Manteniment*). Aquest temps es calcula multiplicant els registres que tenen *Error* e actiu per t.

$$t_e = e \cdot t$$

És important notar que en tots els càlculs hi ha una lleugera imprecisió, provocada principalment per la forma com funciona la informàtica i en especial Python. Tanmateix, en tots els càlculs es considera que aquest error és negligible i que no afecta el resultat final. Finalment, el càlcul de la disponibilitat queda així:

$$\text{Disponibilitat} = \frac{t_w - t_b - t_e}{t_w - t_b}$$

I com que es vol en percentatge es multiplica per 100.

$$\text{Disponibilitat} = \frac{t_w - t_b - t_e}{t_w - t_b} \cdot 100$$

3.1.2. Rendiment

És el resultat de dividir les peces fabricades p_t per la quantitat teòrica que s'hauria d'haver fabricat. El màxim teòric de producció de la fàbrica és el resultat de dividir el temps treballat t_w pel temps de cicle t_c de la línia de producció. Aquest últim és un factor ideal que és definit pel client.

$$\text{Rendiment} = \frac{p_t}{\frac{t_w}{t_c}} = \frac{p_t \cdot t_c}{t_w} \cdot 100$$

3.1.3. Qualitat

Es divideix el nombre de peces «Ok» p_o entre el total de peces fabricades p_t — «Ok» i «Nok». Es considera que una peça és «Ok» quan compleix tots els estàndard de qualitat. Si es detecta algun problema en la qualitat del producte, la peça es considera «Nok».

Hi ha vegades que les peces defectuoses es poden tornar a entrar en la línia, cosa que pot provocar que una mateixa peça estigui entrada dues vegades a la base de dades (una com a peça defectuosa i l'altra com a peça correctament fabricada per exemple). Això no obstant, en aquest projecte internament s'ha definit que aquest fet manca d'importància i que les peces que es tornen a processar són considerades com a noves peces. Això no hauria de ser correcte, ja que aquest índex només accepta com a peces bones o «Ok» aquelles que ho són la primera vegada, fet que provoca que les peces correctes però que prèviament s'han rebutjat es descarten en el càlcul.

Aquest càlcul s'ha programat així perquè és un prototip i no disposem de peces numerades per dur el registre, però es té en compte que en un futur s'hagin de descartar aquestes peces afegint un registre `Reworked` a la base de dades que haurà de valdre `True` si la peça ja ha entrat a la base de dades. Amb aquesta modificació previsiblement es pot crear una consulta que només consideri les peces fabricades per primer cop.

$$\text{Qualitat} = \frac{p_o}{p_t} \cdot 100$$

3.1.4. Eficiència general dels equips

Amb tots aquests paràmetres calculats, l'eficiència general dels equips o OEE per abreviar és el resultat de fer el producte de tots aquests percentatges. Així, el càlcul final és tan fàcil com fer:

$$\text{OEE} = (\frac{\text{Disponibilitat}}{100} \cdot \frac{\text{Productivitat}}{100} \cdot \frac{\text{Qualitat}}{100}) \cdot 100 = \frac{\text{Disponibilitat} \cdot \text{Productivitat} \cdot \text{Qualitat}}{10000}$$

3.1.5. Actualització

Això no obstant, aquest disseny presenta alguns inconvenients importants. Per exemple, podria ser el cas que un manteniment no planificat es dugui a terme enmig d'un descans, o que durant aquest s'activi el bit d'error del PLC per accident. Un altre cas indesitjat seria que hi hagués un descans enmig d'un manteniment. Per evitar que aquests i altres casos puntuals, però plausibles, donin valors incorrectes o extravagants s'ha decidit replantejar aquests càlculs.

Per exemple s'ha decidit canviar els càlculs de t_b. El primer que s'ha fet ha sigut ampliar la taula `table_shifts` de la base de dades — capítol 4 — afegint quatre columnes que contenen l'hora d'inici i durada de les parades planificades i dels manteniments programats. Per convenció, en un torn només hi poden haver una parada planificada i un manteniment programat, tot i que si el futur client així ho vol aquests es poden ampliar segons convingui. D'aquesta forma el càlcul de t_b només es du a terme si l'hora actual coincideix o és superior a l'hora de la parada planificada o del manteniment programat guardats en la base de dades. Si això no s'ha especificat, el càlcul no es realitzarà. Com que en la base de dades també es guarda la seva durada d — en minuts—, el càlcul es simplifica bastant perquè ara només s'han de comptar el nombre k de minuts que han passat.

$$t_b = t_r + t_m$$

On al seu torn t_r i t_m es calculen de la mateixa manera, i només si es compleixen les condicions següents:

- Si l'hora actual és més gran o igual que l'hora d'inici de la parada planificada o del manteniment.
- Si encara no s'ha acabat el temps disponible per la parada o el manteniment. Això s'explica amb més detall en el capítol 6, però resumint una variable interna llegeix la dada de la base de dades, la qual es va restant minut a minut.

$$k_r += 1 \qquad d_r -= 1 \qquad t_r = k_r \cdot t$$

$$k_m += 1 \qquad d_m -= 1 \qquad t_m = k_m \cdot t$$

Per dur a terme això es pressuposa que el flux del programa no pot acabar mai. D'altra banda, cada vegada que es calcula el temps de parada planificada o el temps de manteniment programat s'actualitza el registre corresponent de la taula `table_plc` de la base de dades. D'aquesta forma hi ha la certesa que només es comptin les peces fabricades en el període de funcionament de la màquina, i que ni la qualitat ni la productivitat es vegin penalitzades perquè hi hagi hagut algun error, manteniment o parada en la producció.

Amb aquests canvis, el càlcul del rendiment també canvia, ja que ara només compta les peces que s'hagin produït fora de t_b o de t_e. Així mateix, només es té en compte el període de treball de la màquina:

$$\text{Rendiment} = \frac{p_t}{\frac{t_w - t_b - t_e}{t_c}} = \frac{p_t \cdot t_c}{t_w - t_b - t_e}$$

El canvi en la qualitat només es du a terme afegint condicions a la consulta de la base de dades. És per aquest motiu que el càlcul no varia respecte del plantejament inicial.

3.2. Requeriments

Aquest projecte pretén ser un afegit a les màquines dissenyades per Ingimec. Es vol dotar d'un sistema amigable i altament configurable que sigui un incentiu i un punt diferencial per als potencials clients. Per fer-ho es fa ús de diferents eines que s'han après durant el grau, com el llenguatge de programació Python o l'ús de Bases de Dades.

Per aconseguir això és necessari la creació d'una base de dades relacional, una comunicació directa i continua entre el PLC i el PC i un servidor web que pugui ser allotjat en qualsevol entorn.

3.2.1. Base de dades: SQLite3

SQLite és un sistema de gestió de bases de dades relacional continguda en una petita biblioteca escrita en C. El fet que estigui programat en C li dota de flexibilitat i gran velocitat. El fet que no sigui un procés independent, sinó que sigui una part integral del mateix — en altres paraules, que no calgui definir usuaris per al seu ús — fa que es pugui implementar ràpidament amb independència de per a qui s'hagi creat. La versió 3 permet bases de dades de fins a 2 Terabytes, així com la inclusió de dades de tipus BLOB. Els BLOB — acrònim de *Binary Large Objects* — són elements emprats en bases de dades que serveixen per emmagatzemar dades de gran mida que canvien de forma dinàmica. Aquestes dades sovint solen ser imatges, sons o altres objectes multimèdia.

3.2.2. Comunciacions plc-pc: Snap7

Snap7 és una suite de comunicacions Ethernet multiplataforma de codi obert per la connexió amb PLC de la marca Siemens. El fet que sigui multiplataforma facilita la comunicació dels PLC d'aquesta marca amb múltiples plataformes, i que sigui de codi obert permet la seva aplicació a la indústria perquè no es requereixen llicències privatives que encararien el producte. Snap7 també permet programar les comunicacions amb els PLC a través de diferents llenguatges. Això fa que sigui versàtil en múltiples entorns de programació.

3.2.3. Entorn web: Flask

Es vol crear un sistema robust i funcional, amb una gran flexibilitat i facilitat d'ús. L'entorn de treball en miniatura Flask està basat en Python i permet la creació d'aplicacions web amb un mínim de línies de codi.

Python és un llenguatge de programació d'alt nivell i interpretat, creat per Guido van Rossum a finals dels anys vuitanta. Aquest llenguatge és multiplataforma i suporta diferents paradigmes de programació, com ara la programació orientada a objectes, la programació imperativa o la funcional.

Flask és un entorn de treball en miniatura basat en Python per al desenvolupament web. Aquesta eina és àmpliament emprada pels desenvolupadors que volen crear aplicacions web ràpides i senzilles, ja que proporciona eines i llibreries per gestionar les peticions HTTP, crear plantilles HTML, gestionar sessions i crear bases de dades, entre altres funcionalitats. A més, Flask és modular, el que vol dir que els desenvolupadors poden afegir funcions addicionals a la seva aplicació web a mesura que les necessitin.

3.3. Competència

La necessitat de calcular l'OEE és vital per a les empreses i les fabriques, ja que proporciona una visió detallada de l'eficiència de la producció. També ajuda a millorar les preses de decisions, perquè augmenta la informació que es té sobre la planta i permet una visió més detallada dels punts que s'han de millorar per augmentar la producció. En mesurar l'eficiència de la màquina a través de la disponibilitat, l'eficiència i la qualitat, els gerents i enginyers de planta poden identificar més ràpidament els problemes que estan afectant la producció i prendre mesures per corregir-los. Al mateix temps, també ajuda a determinar si la màquina està produint correctament o a plena capacitat o si, en canvi, s'ha d'invertir en millores o maquinària nova.

És per això que molts fabricants estan invertint a afegir a les seves màquines sistemes de càlcul i visualització de l'OEE. Els fabricants de maquinària estan competint per una quota de mercat cada vegada més petita, de forma que busquen diferenciar-se de la competència implementant sistemes integrats als seus productes. Aquests solen ser atractius i intuïtius perquè hi hagi una bona acollida entre els potencials clients, ja que un sistema difícil de fer servir o de configurar els pot allunyar.

Un altre factor important a l'hora de dissenyar aquests tipus de sistemes és la creixent importància que el càlcul analític té a la indústria. Amb aquests sistemes ja no només es busca visualitzar la informació d'una forma elegant i simple, sinó que també han de ser capaços de mostrar, analitzar i calcular grans quantitats de dades i de càlculs en temps real perquè els gerents puguin prendre les decisions més adequades per millorar la producció.

Per tots aquests motius nombrosos fabricants estan oferint aquests tipus de solucions. Parsec Automation Corp ofereix una solució programari coneguda com a TrakSYS. Aquesta eina proporciona en temps real una visió dels indicadors clau del rendiment de la producció. Per altra banda, Schneider Electric ofereix Wonderware MES Performance Software, que ofereix sistemes de visualització en temps real de la productivitat i la fabricació en múltiples línies de producció, a més de proporcionar informació de diagnòstic i altres paràmetres. Altres empreses que proporcionen sistemes similars són Rockweel Automation amb FactoryTalk Metrics o OEESystems amb PerformOEE. Això no obstant, cap d'aquestes empreses ofereix productes clau en mà, i els fabricants que sí que ofereixen aquests tipus de producte solen subcontractar aquesta part, de forma que actualment no hi ha una competència en aquest sentit.

4. Arquitectura de la Base de dades

La gestió i anàlisis de les dades que es necessiten per al càlcul de l'OEE i d'altres paràmetres no és senzilla, i requereix un raonament i un disseny previ abans de dur a terme qualsevol tasca. Per això, el disseny d'una base de dades robusta i fàcil d'utilitzar és una part fonamental d'aquest treball.

Per això, s'ha decidit implementar una base de dades única per a tot el projecte, que ha de contenir tant les dades llegides des del PLC com els càlculs propis de l'OEE duts a terme. El servidor Flask que ha de fer de visualitzador de les dades a través d'una pàgina web minimalista també ha d'obtenir la informació necessària des d'aquí.

D'aquesta forma s'ha decidit que la base de dades tingui tres taules, una per a cada necessitat de l'aplicació. Primer de tot, és necessari guardar les dades que es van llegint des del PLC; una altra taula ha de servir per emmagatzemar els torns de la línia de producció; per últim, una tercera taula s'ha d'ocupar de guardar les dades calculades per assolir aquest índex. Aquesta última taula també ha de ser consultada pel servidor que ha de mostrar aquestes dades a través d'una web minimalista:

- `table_plc`: guarda les dades enviades pel PLC.
- `table_shifts`: guarda els torns que ha definit prèviament el client.
- `table_oee`: emmagatzema les dades que es calculen en el procés d'obtenció de l'OEE.

4.1. `table_plc`

Nom	Tipus	CP	NULL	Autoincrement	Únic	Valor per defecte
Id	INT	YES	NO	NO	YES	-
Date	TIME	NO	NO	NO	NO	-
Hour	TIME	NO	NO	NO	NO	-
Auto	INT	NO	YES	NO	NO	NULL
Manual	INT	NO	YES	NO	NO	NULL
Audit	INT	NO	YES	NO	NO	NULL
Error	INT	NO	YES	NO	NO	NULL
Maintenance	INT	NO	YES	NO	NO	NULL
OK	INT	NO	YES	NO	NO	NULL
NOK	INT	NO	YES	NO	NO	NULL
Error_codes	TEXT	NO	YES	NO	NO	NULL
Break	INT	NO	YES	NO	NO	0

Taula 4.1.: Taula que mostra les dades que conté la taula de la base de dades encarregada d'emmagatzemar les dades llegides del PLC.

Com ja s'ha dit, gestiona les dades llegides des del PLC. Està conformada pels següents registres (Taula 4.1).

`id` és una columna identificador únic que es genera automàticament — autoincrement — per a cada fila que s'afegeix a la taula. És la clau principal d'aquesta, el que vol dir que no es pot afegir una fila que tingui el mateix valor en aquesta columna que una fila que ja existeixi a la taula.

Els registres `Date` i `Hour` són de tipus `TIME`. Guarden la data i l'hora en què s'enregistra la informació.

`Auto`, `Manual`, `Audit`, `Maintenance` són columnes que representen els bits que envia el PLC. Aquests mostren l'estat de funcionament de la màquina i són booleans, on `True` significa que està funcionant en aquest estat. Si tots els valors són `False`, vol dir que la màquina no està en funcionament.

La columna `Error` mostra en un valor booleà l'estat del bit específic que envia el PLC si s'ha experimentat un error intern. En aquest cas, el programa ha de llegir el DB 99 i escriure el codi d'error que ha passat en la columna `Error_codes`. `Error_codes` és una columna de tipus `TEXT` que conté una llista de tots els codis d'error que hagin pogut passar.

Els registres `Ok` i `Nok` corresponen a uns valors enters que funcionen com a comptadors de les peces fabricades. Així, `Ok` mostra les peces fabricades que no presenten cap defecte, i `Nok` mostra les peces rebutjades. En aquest punt, s'ha de tindre present que el temps de cicle que es valora és de deu segons per peça, i nosaltres escrivim a la base de dades una vegada per minut. Aquest últim fet ja es té en compte en el procés d'escriptura a la base de dades.

Finalment, la columna `Break` és l'única que s'escriu en el procés de càlcul de l'OEE. És un booleà que per defecte es defineix com a falç i que només serà verdader en cas que hi hagi una parada planificada.

En aquesta taula no hi ha restriccions per evitar que es repeteixi la informació emmagatzemada en cap columna —a excepció de la clau primària, l'identificador `id`.

4.2. `table_shifts`

Nom	Tipus	CP	NULL	Autoincrement	Únic
Id	INT	YES	NO	YES	YES
Days	STRING	NO	NO	NO	NO
Start_time	TIME	NO	NO	NO	NO
End_time	TIME	NO	YES	NO	NO
Break_time	TIME	NO	YES	NO	NO
Break_duration	INT	NO	YES	NO	NO
Maintenance_time	TIME	NO	YES	NO	NO
Maintenance_duration	INT	NO	YES	NO	NO

Taula 4.2.: Taula que guarda les dades dels torns, per tal de poder calcular l'OEE correctament.

Emmagatzema els torns de la fàbrica, amb les seves hores i els dies que comprèn, per poder fer els càlculs de l'OEE correctament. Conté informació dels dies, les hores i els temps de descans o de manteniment programats (Taula 4.2).

Com en el cas de `table_plc`, `id` és una columna identificadora que es genera automàticament. És única i té «autoincrement» per cada fila que s'afegeix a la taula. S'usa com a clau principal

d'aquesta.

Els horaris dels torns es guarden en dues columnes separades, on `Days` és un `string` que guarda els dies de la setmana on aquell torn és vàlid. `Start_time` i `End_time` guarden l'hora d'inici i final d'aquest i són de tipus `time`. Tant aquestes columnes com l'identificador no poden ser nuls.

Per poder calcular la disponibilitat correctament, necessitem saber els temps de parades planificades. Per recordar la forma de calcular-ho es pot consultar la secció 3.1.

Els registres `Break_time` i `Maintenance_time` són valors de tipus `time` que guarden les hores d'inici de les parades planificades o dels manteniments, si n'hi ha. Per altra banda, `Break_duration` i `Maintenance_duration` són valors enters que guarden la durada de la parada planificada en minuts. Com que pot ser que un torn no tingui una parada planificada ni cap manteniment — torns que per exemple no arribin a una hora per algun motiu —, aquests valors poden ser nuls.

4.3. `table_oee`

Nom	Tipus	CP	NULL	Autoincrement	Únic
Id	INT	YES	NO	YES	YES
Date	TIME	NO	NO	NO	NO
Hour	TIME	NO	NO	NO	NO
Oee	REAL	NO	YES	NO	NO
Availability	REAL	NO	YES	NO	NO
Performance	REAL	NO	YES	NO	NO
Quality	REAL	NO	YES	NO	NO
Ok	INT	NO	YES	NO	NO
Nok	INT	NO	YES	NO	NO
Theoretical_production	INT	NO	YES	NO	NO
Work_time	INT	NO	YES	NO	NO
Rest_time	INT	NO	YES	NO	NO
Break_time	INT	NO	YES	NO	NO
Maintenance_time	INT	NO	YES	NO	NO
Error_time	INT	NO	YES	NO	NO

Taula 4.3.: Taula que guarda els valors calculats per obtindre l'OEE, així com els seus paràmetres i el seu valor final.

Potser la més important de totes, ja que és l'encarregada d'emmagatzemar tots els càlculs i tota la traçabilitat que posteriorment es mostrarà en la pàgina web. També és la taula que conté més registres, com es pot veure a la Taula 4.3.

Com en les taules anteriors, la clau primària és un identificador `id` enter que es va incrementant a mesura que es van escrivint registres. Igual que en les taules anteriors, aquest registre es fa servir com a clau primària de la taula.

`Date` i `Hour` són de tipus `time` i guarden la data i l'hora en què s'ha enregistrat la informació. Tant aquests dos registres com l'identificador `id` no poden ser valors nuls.

El valor de l'OEE calculat, així com dels seus paràmetres, es guarden a les columnes que tenen el mateix nom. Són valors de tipus `real`, poden ser nuls i mostren la informació en

forma de percentatge, amb dos decimals. Així mateix, i com el seu nom indica, les columnes `Ok` i `Nok` són valors enters que emmagatzemen, respectivament, el total de les peces bones i dolentes generades d'ençà que el torn ha començat. La columna `Theoretical_production` guarda les peces que segons el temps de cicle s'haurien d'haver fabricat en el mateix període de temps.

Finalment, `Work_time`, `Break_time`, `Maintenance_time` i `Error_time` són de tipus enter, i guarden els temps de funcionament, de parada planificada, de les parades planificades per manteniments i dels errors que hi ha hagut en un torn.

En aquesta taula, tant els valors que representen l'OEE i els seus paràmetres com els valors generats per fer aquest càlcul poden ser nuls, així com no únics.

5. Comunicació amb el plc i escriptura de la base de dades

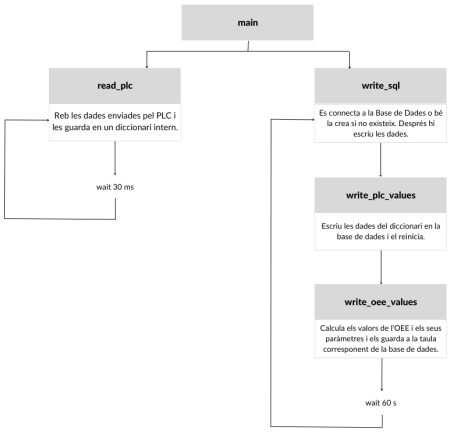

Figura 5.1.: Esquema del funcionament del programa principal.

S'ha decidit programar les comunicacions amb el PLC i els càlculs de l'efectivitat total dels equips en un mateix mòdul. S'ha programat així perquè es fa servir la mateixa base de dades per a tots dos objectius.

D'aquesta forma, s'ha creat un programa Python amb dos fils d'execució. Un fil en Python — també conegut com a *thread* — és una seqüència d'execució independent dins un programa que pot executar diverses tasques alhora. Així, un programa que utilitza fils té la capacitat de dur a terme diverses tasques al mateix temps, per la qual cosa permet millorar el rendiment i l'eficiència. En Python, els fils es poden crear mitjançant el mòdul `threading`. Cada fil té la seva pròpia pila d'execució, per la qual cosa són totalment independents del fil principal del programa.

S'ha de tindre en compte, però, que en alguns casos l'ús de fils mal planificat pot produir errors, com condicions de carrera o bloquejos. Una condició de carrera és un problema que ve provocat quan dos o més fils poden intentar accedir i modificar a la vegada un mateix recurs. Per exemple, un exemple de condició de carrera seria que un fil estigués llegint el valor d'una variable mentre un segon fil l'està modificant. Això provocaria que el primer fil llegís un valor incorrecte o inconsistent. Els bloquejos mutus o *deadlock* es produeix en el moment que dos o més fils estan esperant que l'alliberament de recursos utilitzats per algun altre fil o programa. Això provoca que els fils es bloquegin mútuament i no puguin avançar, provocant un estancament del programa. S'ha de tindre compte amb els *deadlocks* perquè sovint són difícils de detectar i solucionar, ja que solen ser aleatoris i poden provocar problemes diferents.

Tenint en compte això, s'ha creat un programa amb dos fils diferents d'execució — Imatge 5.1. El primer fil s'encarrega de gestionar les comunicacions amb el PLC i guardar la informació

rebuda en un diccionari. El segon fil gestiona la base de dades, hi escriu la informació rebuda des del PLC i els resultats que s'han calculat per obtenir l'OEE:

- `read_plc`: gestiona les comunicacions amb el PLC i guarda les dades llegides en un diccionari que serà consultat posteriorment pel segon fil. Aquest fil s'executa en un bucle sense fi cada 30,00 ms.
- `write_sql`: llegeix les dades del diccionari i les guarda en la taula `table_plc` de la base de dades. Posteriorment, crida la classe `oee` per calcular l'eficiència general dels equips i guarda els resultats en la taula `table_oee` de la base de dades. Tot això és fa en un bucle infinit una vegada per minut.

Tots els codis així com els documents d'aquest treball estan recollits en el dipòsit `6q4598/TFG` de GitHub. La seva estructura està documentada tant en un arxiu `Readme` dins del propi dipòsit com en l'apèndix A.

5.1. Programa principal

La funció principal del programa és ben senzilla. La seva única finalitat és crear i executar els dos fils.

```
def main():
    read_plc_thread = Thread(target = read_plc)
    write_sql_thread = Timer(10, function = write_sql)
    read_plc_thread.start()
    write_sql_thread.start()
```

5.2. `read_plc`

Aquest fil llegeix les dades del PLC i les guarda en un diccionari compartit amb el fil `write_sql`. És infinit, i es va executant recursivament cada 30,00 ms. Per tal de comprovar la robustesa del codi i veure que es llegeixen les dades correctament, s'ha programat un PLC amb un banc de proves.

Figura 5.2.: El banc de probes conté un PLC Siemens SIMATIC S7-1200 amb una botonera de control i un HMI.

5.2.1. plc i banc de proves

Per tal de provar les comunicacions del PLC i la correcta lectura dels *data block* — d'ara endavant DB — d'aquest, s'ha programat un banc de proves amb un PLC Siemens SIMATIC S7-1200 — Figura 5.3. En aquest PLC s'han definit tres DB, dels quals se n'han fet servir dos. Els DB que s'han definit són el 91, el 92 i el 100. D'aquests, el 91 gestiona les dades que s'han de llegir des de la Rapsberry PI, el 92 es guarda per si en algun moment es volen enviar dades des de la Raspberry PI al PLC, i el 100 guarda una llista de tots els possibles errors que hi poden haver, que s'enviaran si hi ha hagut algun problema amb el PLC i s'ha activat el bit d'error del DB 91.

db 91

En el dB 91 s'han definit els següents bits, com es pot veure en la taula 5.2.1.

Descripció	Tipus	Byte	Bit
Automàtic	Bool	0	0
Manual	Bool	0	1
Auditoria	Bool	0	2
Error	Bool	0	3
Manteniment	Bool	0	4
Peça OK	Bool	1	0
Peça NOK	Bool	1	1

Taula 5.1.: Taula que mostra els bits definits en el DB 91.

Els cinc primers bits corresponen al mode de funcionament del PLC, que poden ser:

- Automàtic
- Manual
- Auditoria
- Error
- Manteniment

Només en el primer cas es fa el càlcul de l'OEE, mentre que els modes de *manual* i *auditoria* s'ignoren. El bit d'*error* s'activa quan hi ha hagut algun error en la màquina, instant en el que també s'envia el codi de l'error mitjançant el DB 100. El bit de *manteniment* s'ha d'activar quan hi ha algun manteniment a la màquina no planificat. Tant si el bit d'*error* com si el bit de *manteniment* estan actius, l'OEE ho ha de gestionar i contemplar en la disponibilitat com a t_e.

Els bits de *peça ok* i *peça nok* s'activen quan s'ha fabricat una peça i aquesta es dona com a bona o dolenta, respectivament. Com que el programa llegeix el DB cada 30,00 ms, aquesta lectura s'ha de gestionar correctament si es volen llegir totes les peces. Per això, el PLC activa aquests bits a 1 durant una finestra de 100,00 ms per donar temps a la Raspberry PI de llegir-ho correctament — posteriorment en la secció 5.2.2 s'explicarà amb detall com es gestiona.

db 100

Aquest DB només enviarà el codi d'error corresponent a la Raspberry PI en cas que el bit d'*error* del DB 91 estigui actiu. En l'apèndix B hi ha la llista de tots els errors que s'han contemplat. Tanmateix, i a causa de a la limitació que ofereix el banc de proves, l'únic error que de veritat es té en compte és el primer, tots els altres s'han definit segons el que es va decidir en el seu moment aprofitant programes previs d'Ingimec, però no es fan servir. Això és així perquè tots els polsadors i selectors disponibles, així com l'HMI, es fan servir per altres accions.

Així, els errors que s'han activat en el DB 100 s'afegeixen en una llista dins el diccionari — secció 5.2.2. D'aquesta forma es poden afegir a la taula indicada de la base de dades.

Banc de proves

S'ha definit un banc de proves que mitjançant botons i interruptors simula ser una màquina real. El banc de proves que s'ha creat consta de les següents parts:

- Dos interruptors selectors de dues posicions amb retorn, els quals gestionen el mode — *automàtic*, *manual*, *auditoria* o *error* que es vol simular.
- Dos polsadors que simulen ser les peces creades. N'hi ha un per les peces «bones» i un altre per peces «dolentes».
- Un panell HMI Siemens KTP400 Basic PN, que serveix per simular l'únic mode que no es gestiona amb els selectors: els manteniments no planificats.

Així, el primer interruptor — començant des de l'esquerra — activa el mode *automàtic* si es gira cap a l'esquerra, o *manual* en cas contrari. L'altre interruptor activa els modes d'*auditoria* o d'*error*.

D'altra banda, en prémer el primer polsador — el de l'esquerra —, s'envia una seqüència que activa el bit de *peça ok* durant 100,00 ms cada 10,00 s. Quan es manté polsat un segon o més, la seqüència s'atura. El funcionament és el mateix per l'altre polsador, tot i que aquest activa el bit de peça dolenta. Aquest sistema fa que no hàgim de clicar el botó un nombre elevat de peces per a obtenir valors de referència adequats per al càlcul de l'OEE.

Com que en un inici no es va tindre en compte que hi poguessin haver manteniments no planificats, i ja que en el banc de proves no hi havia espai per afegir aquesta opció, es va programar un *hmi* per tal de poder-ho gestionar. Aquest disposa de dos botons virtuals, els quals serveixen per activar o desactivar aquest mode, a més d'un indicador visual que serveix per saber si el mode està activat o no.

S'ha de dir, però, que quan es va programar aquest banc de proves no es van tindre en compte certes incongruències. Per exemple, es pot donar el cas que els dos selectors estiguin fora de la posició neutral, de forma que dos modes poden estar actius al mateix temps. Tampoc es va contemplar que no es poguessin enviar peces en els modes de *manteniment*, *auditoria* o *error*. Malgrat això, i com que en la vida real això no podria passar, no es van tindre en compte aquests detalls i es va limitar a parar compte a l'hora de manipular el banc.

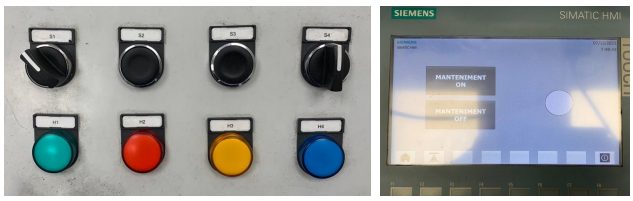

Figura 5.3.: Detall de la botonera i del HMI del banc.

5.2.2. Codi

Per tal d'activar les comunicacions amb el PLC i poder llegir dades s'ha creat un programa Python. S'ha triat aquest llenguatge perquè és fàcil i ràpid de programar, permet crear programes amb poques línies de codi i té múltiples llibreries i paquets que faciliten la programació.

D'aquesta forma, per tal de poder comunicar la Raspberry PI amb el PLC, s'ha fet servir el paquet `python-snap7`. Aquest paquet permet les comunicacions amb els PLC de la marca Siemens.

Python-snap7

Per al tema de les comunicacions, s'ha creat un programa Python que es comunica amb el PLC a través de Snap7. Snap7 és un paquet de comunicació Ethernet multiplataforma. Està disponible tant per arquitectures de 32 com de 64 bits i compta amb suport per múltiples tipus de processadors. De codi obert, permet interactuar de forma nativa amb els PLC del fabricant Siemens [**siemens:s7**]. Tot i que ha estat dissenyat per superar les limitacions pròpies dels sistemes OPC per a transferències de gran quantitat de dades, aquesta suite s'adapta bé a petites trameses de dades com les que s'han dut a terme. Es pot trobar més documentació sobre Snap7 a [Nar].

Aquesta llibreria permet tant la lectura com l'escriptura dels PLC de la sèrie 7 de Siemens. El fet que estigui basada en `ctypes` permet accés directe a qualsevol llibreria compartida amb el sistema operatiu. Per això es pot deixar de fer `wrapper` manualment en els programes, ja que aquesta tecnologia permet escriure codi en Python pur. Es pot veure més informació i exemples de com comunicar Python amb C mitjançant aquesta tecnologia a la pàgina oficial del projecte [GS13].

Per instal · lar aquest `wrapper` hem de fer-ho mitjançant `pip`.

```
pip install python-snap7
```

Diccionari

A causa del fet que el fil `read_plc` s'executa cada 30,00 ms, i a que el fil `write_sql` ho fa a una freqüència d'una vegada per minut, s'ha hagut de trobar una forma d'emmagatzemar les dades llegides del PLC. Així, s'ha creat un diccionari que emmagatzema les dades llegides per posteriorment escriure-ho a la base de dades. Un cop guardada tota la informació, és borra tot el contingut del diccionari i es torna a començar tot el procés. La forma com és borra el diccionari i es guarden les dades s'explica amb més detall a la secció 5.3. El diccionari que gestiona tota la informació que es llegeix del DB 91 és el que segueix, amb les següents parelles de clau/valor:

```
db_values = {'Auto': 0,
    'Man': 0,
    'Audit': 0,
    'Error': 0,
    'Maintenance': 0,
    'Ok': 0,
    'Nok': 0,
    'ErrorCodes': []}
```

- L'enter `Auto` defineix si el mode en automàtic és actiu.
- L'enter `Man` defineix si el mode manual és actiu.
- L'enter `Audit` defineix si el mode d'auditoria és actiu.
- L'enter `Error` defineix si hi ha algun error en el PLC.
- L'enter `Maitenance` defineix si s'està fent un manteniment no programat.
- L'enter `Ok` guarda el nombre de peces bones que s'han fabricat.
- L'enter `Nok` guarda el nombre de peces defectuoses que s'han fabricat.
- La llista `ErrorCodes` conté els errors que s'hagin pogut produir.

Càlcul de les peces fabricades

El programa llegeix les dades del PLC cada 30,00 ms, com ja s'ha dit. Aquesta decisió comporta un error, i és que si s'ha fabricat una peça al cap de 29,00 ms d'haver llegit les dades o si, en canvi, s'ha fabricat després de 31,00 ms, aquesta no es llegirà. Per arreglar això s'ha fet una finestra, de forma que tant el bit de peça correcta com el bit de peça dolenta es mantenen en estat alt durant un temps prou llarg.

Durant les proves, s'ha comprovat que una finestra de 100,00 ms és suficient per llegir la peça correctament, tot i que s'hauran de dur a terme les mesures necessàries en el codi per evitar que la mateixa peça es llegeixi diversos cops. Un comptador emmagatzemat dins del diccionari s'encarrega de guardar el nombre de peces fabricades per poder introduir-les en la base de dades quan sigui el moment.

5.2.3. Fil d'execució

El codi següent mostra tot el fil complet de lectura dels DB 91 i 99. Aquesta funció conté un bucle infinit que s'executa cada 30,00 ms. La primera línia d'aquest crea un objecte client del paquet `snap7` que representa una connexió al PLC. Així mateix, també es fa servir per llegir-lo o interactuar-hi. Seguidament, es comprova si hi ha una connexió activa mitjançant el mètode `get_connected()`. Aquest és un mètode de `snap7` que retorna `True` en cas que hi hagi una connexió activa, o `False` en cas contrari. En qualsevol dels dos casos imprimeix informació sobre l'estat de la connexió.

Si hi ha una connexió activa amb el PLC, activa el *flag* `plc_connected` per tal d'informar el fil `write_sql` que pot escriure a la base de dades. Després, comença la lectura del DB 91, que es guarda en la variable `read_plc`. El mètode propi `convert_byte_to_bit()` converteix

la informació llegida en un array de bits, i ho guarda en la variable `read_plc_bin` per tal de fer més fàcil la inclusió de tota la informació llegida en el diccionari. Aquesta es guarda segons l'especificat en la subsecció 5.2.1.

Aquí s'ha de fer un incís en la forma de comptar les peces. Com ja s'ha dit anteriorment, quan s'activen els bits de *peça ok* i *peça nok* ho fan durant un instant de temps concret. Per comptar això, és crida al mètode `count_pieces()`. Aquest mètode accepta dos paràmetres, corresponents als bits de *peça ok* i *peça nok*. Els paràmetres `piece_ok` i `piece_nok` corresponen al valor d'aquests bits. Quan hi ha un flanc de pujada — el paràmetre és «1» —, activa el *flag* corresponent i s'incrementa una unitat el comptador de peces. Si hi ha un flanc de baixada — el valor és «0» —, el flag corresponent es desactiva. El recompte de peces es du a terme mitjançant les variables `num_ok` i `num_nok`. L'ús de *flags* permet saber si anteriorment ja hi ha hagut un flanc de pujada del bit per tal de no comptar més d'una vegada la mateixa peça. D'aquesta forma, en les dues primeres condicions s'activa el *flag* si aquest està desactivat i hi ha hagut un flanc de pujada al mateix temps que es compta una peça. En les últimes dos, es desactiva el *flag* corresponent si hi ha hagut un flanc de baixada i aquest està actiu. Tots els altres casos possibles s'ignoren.

Finalment, si hi ha hagut una excepció durant aquest procés, s'imprimeix un missatge d'error i es torna a executar el bucle. Així hi ha la certesa que el programa no acabi abruptament.

```
def read_plc():
    """
    Read DB 91 and 100 from the PLC.
    """
    global plc_connected, num_ok, num_nok, fabricated_ok, fabricated_nok

    while True:
        client = snap7.client.Client()

        try:
            client.connect('192.168.0.1', 0, 0)

            # If client is connected to the PLC, read DB and converts it to a
                bit array.
            if (client.get_connected()):

                if plc_connected == False:
                    print("PLC connected.")

                plc_connected = True
                read_plc = client.db_read(91, 0, 2)
                read_plc_bin = convert_byte_to_bit(read_plc)

                count_pieces(int(read_plc_bin[10]), int(read_plc_bin[9]))
                # Then, count pieces and write it to the dictionary «db_values».
                db_values.update({'Auto': int(read_plc_bin[-1]), 'Man':
                    int(read_plc_bin[-2]), 'Audit': int(read_plc_bin[-3]),
                    'Error': int(read_plc_bin[-4]), 'Maintenance':
                    int(read_plc_bin[-5]), 'Ok': num_ok, 'Nok': num_nok,
                    'ErrorCodes': ''})

            else:
                print("PLC disconnected. Trying to reconect")
```

```
        except Exception as e:
            print("Error connection PLC. Error: ", e)
            continue

        # Set a runtime delay.
        time.sleep(PLC_SLEEP)
```

5.3. `write_sql`

El segon fil del programa guarda els valors llegits del PLC en la base de dades definida en el capítol 4. Després, calcula l'OEE i guarda els resultats en la taula corresponent.

La primera acció del fil és comprovar que existeixi la base de dades. Si aquesta existeix, s'estableix una connexió utilitzant la funció `connect()` del mòdul `sqlite3`. `sqlite3` és un mòdul que proporciona una API eficient per treballar amb bases de dades relacionals SQLite. En Python, aquest mòdul permet als programadors connectar-se, crear i efectuar operacions tant d'escriptura com de lectura amb aquests tipus de bases de dades. En cas que la base de dades no existeixi, la crea i crida la funció `create_tables()`. Aquesta funció crea les taules segons el que s'ha especificat anteriorment. Tant en el primer com en aquest cas, així com si hi ha hagut alguna excepció, s'imprimeix un missatge informatiu.

Un cop s'ha connectat a la base de dades i s'han creat — si fa falta — les taules corresponents, es comença el bucle infinit. Aquest s'executa una vegada per minut, i només ho fa en cas que la connexió amb el PLC s'hagi establert correctament. En cas afirmatiu, es crida la funció `write_plc_values()`, que accepta com a paràmetres la connexió i el cursor a la base de dades. Aquest mètode copia el contingut del diccionari `db_values` i dels comptadors de les peces `num_ok` i `num_nok` en llurs variables locals. Seguidament, reinicia el contingut de `db_values` així com dels comptadors. Finalment, ho escriu a la base de dades. Aquesta funció s'ha dissenyat així per no bloquejar el diccionari o els comptadors en el procés d'escriptura a la taula, i permetre que el fil `read_plc` no s'hagi d'esperar que aquest procés acabi.

Posteriorment, es comprova mitjançant un *flag* si s'ha creat una instància de la classe `oee`. Aquesta classe calcula els valors de l'OEE, com s'explica més endavant en el capítol 6. Si el *flag* corresponent no s'ha activat, l'activa i crea una instància d'aquesta classe. Tot seguit inicialitza els següents valors mitjançant la invocació dels mètodes corresponents:

- El mètode `get_start_shift_time()` obté l'hora d'inici del torn segons el dia i hora actuals. Això ho fa consultant a la taula `table_shifts` de la base de dades.
- D'igual forma, el mètode `get_end_shfit_time()` fa el mateix per a l'hora final.
- Els mètodes `get_break_time()` i `get_maintenance_time()` aconsegueixen l'hora d'inici del temps de descans i dels manteniments planificats. En cas que no hi hagi un temps de descans o un manteniment planificats, aquests mètodes retornen -1.
- En últim lloc, els mètodes `get_break_duration()` i `get_maintenance_duration()` assoleixen la durada de les parades planificades o dels manteniments programats, respectivament. D'igual forma que en els dos mètodes anteriors, si no s'han planificat parades o manteniments aquestes funciones retornen -1.

Un cop creada la instància a aquesta classe i inicialitzats els valors, es crida el mètode `write_oee_values()` que accepta com a paràmetres la connexió a la base de dades, el cursor, la instància a la classe `oee` i un *flag* que s'activa si hi ha hagut o no un error. D'aquesta forma, i com s'explica en el capítol 6, en cas que hi hagi algun error en el PLC s'ignoren tant els càlculs de la productivitat com de la qualitat. Aquesta funció crida els mètodes adequats de la classe per tal de realitzar els càlculs de l'OEE. Un cop calculats, s'escriuen a la taula corresponent de la base de dades.

```
def write_sql():
    """
    Write in the database the values readed from PLC and saved in the
        dictionary «db_values». Then, reset «db_values» values.
    """
    global num_ok, num_nok, plc_connected, trheads_list
    flag = False

    # Start DB connection. If the database not exists, create tables.
    if (os.path.exists(sql_path)):
        print("Database conected.")
        sql_connection = sqlite3.connect(sql_path)
        sql_cursor = sql_connection.cursor()

    else:

        try:
            print("Creating database.")
            sql_connection = sqlite3.connect(sql_path)
            sql_cursor = sql_connection.cursor()
            create_tables(sql_cursor)

        except sqlite3.OperationalError as e:
            print("Error connecting database. Error: {}\nStopping
                program.".format(e))

    while True:
        if (plc_connected):

            now_error = db_values['Error']

            # Insert PLC and OEE values in the database.
            write_plc_values(sql_connection, sql_cursor)

            if (flag == False):
                # Create OEE class instance.
                flag = True
                current_oee = oee(DB_SLEEP, CYCLE_TIME, sql_connection,
                    sql_cursor)
                current_oee.get_start_shift_time()
                current_oee.get_end_shift_time()
                current_oee.get_break_time()
                current_oee.get_break_duration()
                current_oee.get_maintenance_time()
                current_oee.get_maintenance_duration()

            write_oee_values(sql_connection, sql_cursor, current_oee, now_error)
```

```
time.sleep(DB_SLEEP)
```

Aquí s'ha de fer un incís, ja que els canvis de torn s'han de tractar com correspon. D'aquesta forma, la primera acció que es realitza en el mètode `write_oee_values()` és la comprovació que l'hora actual estigui dins del torn corresponent. Si, en canvi, l'hora actual està fora del torn, es tornen a iniciar els valors anteriors i es tornen a cridar els mètodes corresponents de la classe `oee` explicats en el capítol 6, amb l'objectiu d'obtenir els nous valors per al torn següent. Per exemple, si l'hora actual és més gran que l'hora final del torn o bé — i tot i que això sigui impossible — l'hora actual sigui més petita que l'hora inicial d'aquest, vol dir que s'ha començat un nou torn i que, per tant, torna a cridar els mètodes anteriors per trobar els temps actualitzats. El primer bucle `if` dins del mètode `write_oee_values()` du a terme aquesta acció.

6. Càlcul de l'OEE

La classe `oee` calcula l'efectivitat total dels equips i tots els seus paràmetres. Aquesta classe disposa de les variables i mètodes que es descriuen a continuació. Només és instanciada una vegada en el programa principal.

6.1. Atributs

La classe `oee` disposa de divuit atributs d'instància, dels quals:

- Cinc corresponen a les constants que necessitem per als nostres càlculs. Venen donades generalment pel client i teòricament no es poden canviar.
- Sis corresponen a les dades del torn. S'obtenen directament de la base de dades i es calculen cada vegada que comença un torn nou.
- Quatre per cada un dels paràmetres de l'OEE. Es reinicien cada vegada que comença un torn nou.
- Cinc per les variables necessàries per a aquest càlcul. També s'inicialitzen en començar un nou torn.

6.1.1. Constants

En Python no existeix un tipus de variable anomenat «constant» com en altres llenguatges. Malgrat això, podem aconseguir el mateix efecte definint variables d'instància i nomenant-les en majúscules. D'aquesta forma s'està indicant que el seu valor no hauria de ser modificat.

La classe `oee` conté cinc constants d'instància, les quals s'inicialitzen en el mètode `__init__` de la classe. Aquest mètode accepta quatre paràmetres, corresponents als primers valors que es mostren a continuació. Les constants de la classe `oee` es poden veure a la taula 6.1.

Nom	Tipus	Descripció
`SQL_CONNECTION`	`sqlite3.connect()`	És la connexió a la base de dades.
`SQL_CURSOR`	`sqlite3.cursor()`	Representa el cursor de la base de dades.
`INTERVAL_DURATION`	`int`	Correspon a la durada entre escriptures
`CYCLE_TIME`	`int`	És el temps de cicle de fabricació.
`OBJECT_DB`	`db()`	Objecte de la classe `db`

Taula 6.1.: Taula que relaciona les constants de la classe `oee` amb el seu tipus.

Tant `SQL_CONNECTION` com `SQL_CURSOR` es defineixen en el fil `write_sql`, mentre que `INTERVAL_DURATION` i `CYCLE_TIME` són valors enters que s'ha definit també com constants en el mateix fil. `OBJECT_DB` és un objecte que pertany a la classe `db` i que permet realitzar consultes sense necessitat de redundar en el codi.

6.1.2. Variables

Les variables d'instància són variables que es defineixen dins d'una classe mitjançant el mètode inicialitzador. Solen estar associades amb una instància particular de la classe, per la qual cosa cada instància d'aquesta tindrà la seva pròpia còpia de la variable. Com en el codi aquesta classe només és instanciada una vegada, això no ens és rellevant.

D'aquesta forma en la classe s'han definit en total tretze variables d'instància, de les quals n'hi ha: quatre de tipus `string` corresponents a les hores d'inici i final del torn, manteniments o parades planificades — si n'hi ha —; dos de tipus enter que corresponen a la durada de la parada o manteniment; quatre enters per al càlcul de l'índex OEE i cada un dels seus valors i tres enters més per als càlculs interns. La taula 6.2 mostra un resum d'aquestes variables i el seu tipus.

Nom	Tipus	Descripció
`n_time`	`string`	N'hi ha 4 i guarden l'hora llegida des de la taula `table_shifts`.
`n_duration`	`int`	N'hi ha 2 i guarden el temps de duració llegit de la taula `table_shifts`.
OEE paràmetres	`int`	N'hi ha 4, una per paràmetre calculat.
`current_n`	`int`	N'hi ha tres, i corresponen als valors necessàris per al càlcul de l'OEE.

Taula 6.2.: Taula que relaciona les variables de la classe `oee` amb el seu tipus.

6.2. Mètodes

Aquesta classe conté, sense comptar el mètode constructor, un total de 22 mètodes. Aquests es poden dividir, al seu torn, en tres grans categories: de modificació, de consulta i de càlcul.

Els mètodes de modificació serveixen per inicialitzar variables de la classe i es criden en començar el programa o bé un torn nou. Els mètodes de consulta fan el mateix que els anteriors, però extraient la informació de la base de dades. També solen ser més complicats, perquè han de tractar diverses condicions abans d'executar la consulta. Els mètodes de càlcul, com el seu nom indica, fan les operacions necessàries per obtenir el valor de l'OEE i els seus components.

6.2.1. Mètodes de modificació

Inicialitzen algunes de les variables i constants d'instància de la classe. Aquests són els mètodes més senzills perquè no disposen de condicions, comprovacions ni bucles. Els mètodes de consulta de la classe són:

```
def set_interval_duration(self, INTERVAL_DURATION)
def set_cycle_time(self, CYCLE_TIME)
def reset_values(self)
def close_connection(self)
```

`Set_interval_duration(self, INTERVAL_DURATION)` estableix la constant d'instància `INTERVAL_DURATION` segons el valor que se li hagi passat com a paràmetre. En els fils d'execució no es crida mai, però si en algun moment o en determinats torns ens interessa canviar-la s'ha de fer executant aquest mètode.

`set_cycle_time(self, CYCLE_TIME)` fa el mateix que el mètode anterior. Com aquesta, en el programa principal no es crida mai, però està implementat per cobrir aquells casos futurs on sigui necessari.

`reset_values(self)` es crida cada vegada que comença un torn nou. Mitjançant aquesta funció, les següents variables s'inicialitzen al seu valor per defecte per poder començar des de zero tots els càlculs. D'aquesta forma tenim els càlculs de l'OEE per torns.

`close_connection(self)` està implementada per si en algun moment cal tancar la connexió amb la base de dades.

6.2.2. Mètodes de consulta

Aquests mètodes serveixen, com els del punt anterior, per a modificar els atributs de la classe. No obstant això, la diferència rau en el fet que aquests ho fan consultant a la base de dades. Com que hi han definits tres torns diferents — matí, tarda o nit — s'han de realitzar les consultes correctament. Per exemple, en els torns de matí o de tarda no hi ha problema, ja que transcorren en un mateix dia, però en els torns de nit on l'hora inicial i l'hora final estan en dies diferents fer la mateixa consulta no és possible. En concret, els mètodes de consulta de la classe són els següents:

```
def get_break_time(self)
def get_break_duration(self)
def get_maintenance_time(self)
def get_maintenance_duration(self)
def get_start_shift_time(self)
def get_end_shift_time(self)
def update_break_true(self)
def get_num_error(self)
def total_pieces_fabricated(self)
def total_pieces_ok_fabricated(self)
def total_pieces_fabricated(self)
```

En el programa principal els dos primers mètodes que es criden corresponen a `get_start_shift_time(self)` i `get_end_shift_time(self)`. Aquests dos mètodes obtenen l'hora d'inici i final del torn actual, respectivament. Cada vegada que s'acaba el torn i comença un torn nou es reinicien tots els atributs de la classe mitjançant el mètode `reset_values(self)` i es tornen a aconseguir les hores d'inici i final tornant-los a cridar.

Com que típicament hi ha tres torns — com es pot consultar a la taula 6.3 — les consultes s'han de realitzar correctament, ja que si no s'obtenen resultats inesperats. Per posar un exemple, en els torns de matí o de tarda aquestes hores s'assoleixen extraient de la base de dades el registre que compleixi el dia, i que l'hora actual estigui compresa entre les hores inicials i finals d'aquest. En canvi, si fem el mateix per al torn de nit, hi hauria una incongruència, ja que en la taula no hi ha una `End_time` que sigui estrictament més gran que, per exemple, les 22 : 00 : 00.

Per corregir-ho, tots dos mètodes fan el mateix: obtenen l'hora actual i comproven que aquesta sigui superior o igual a l'hora inicial més petita de la taula i inferior a l'hora final

més gran. Com que aquests són valors que no solen variar i són definits pel client, durant les proves s'ha decidit seguir la taula d'hores següent perquè moltes empreses tenen un horari de producció semblant. Si es donés el cas que un client tingués horaris de producció diferents, s'haurien de modificar aquests paràmetres.

Torn	Hora d'inici	Hora final
Matí	06:00:00	14:00:00
Tarda	14:00:00	22:00:00
Nit	22:00:00	06:00:00

Taula 6.3.: Taula que mostra els torns i les seves hores d'inici i final.

En el codi, això es tradueix en una estructura com la de sota. Per totes dues funcions el codi segueix el mateix esquema. També realitzen les mateixes consultes però canviant el paràmetre a buscar, ja que per obtenir l'hora inicial es busca el valor màxim de la taula mentre que per a l'hora final es busca el valor més petit — això només en els casos en què el torn sigui nocturn.

```
if (current_time >= '06:00:00' and current_time < '22:00:00'):
    sql_query = "SELECT Start_time FROM table_shifts WHERE days LIKE '%{}%' AND
        Start_time <= '{}' AND End_time > '{}'".format(time.strftime("%A"),
        current_time, current_time)
else:
    sql_query = "SELECT MAX(Start_time) FROM table_shifts WHERE days LIKE
        '%{}%'".format(time.strftime("%A"))
```

En sortir de la condició s'executa la consulta a la base de dades mitjançant el mètode `write_to_db()` de la classe `db`. Posteriorment, s'inicialitzen les variables d'instància segons el resultat obtingut i tenint en compte els possibles errors. Finalment es retorna el resultat.

A continuació es criden `get_break_time(self)` i `get_maintenance_time(self)`. Aquests mètodes obtenen l'hora d'inici del descans o del manteniment programat, respectivament. Aquí la consulta és simple, ja que per convenció en un torn com a màxim només hi pot haver un descans i un manteniment. Així, sabent el dia de la setmana i l'hora d'inici del torn la consulta que s'ha de realitzar és ben simple — la consulta que obté l'hora d'inici del manteniment programat és idèntica a aquesta només canviant el registre a obtenir:

```
sql_query = "SELECT Break_time FROM table_shifts WHERE days LIKE '%{}%' AND
    Start_time = '{}'".format(time.strftime("%A", self.start_shift_time)
```

Finalment, en el programa principal els dos últims mètodes que es criden abans de continuar amb l'execució són `get_break_duration(self)` i `get_maintenance_duration(self)`. Tampoc tenen cap misteri, ja que són els mateixos mètodes que abans però obtenint valors enters.

En aquest punt cal comentar que si no s'ha definit cap temps de descans o de manteniment a la base de dades, el mètode `write_to_db` sempre retornarà -1. D'aquesta forma hi ha la certesa que no es retornen valors inesperats o errors, i que sempre podrem tractar aquests en

conseqüència. Els següents mètodes a comentar es criden en el moment de fer els càlculs de l'OEE.

Disponibilitat

Durant el càlcul de la disponibilitat l'única consulta que es realitza és `get_num_error(self)`. Aquí el codi és més complex que en els mètodes anteriors, ja que s'han de distingir tres possibles casos:

- Si l'hora actual és més gran o igual que les 06 : 00 : 00 i més petita que les 22 : 00 : 00 — segons s'observa en la taula 6.3.
- Si l'hora actual és més gran que les 22 : 00 : 00.
- Si l'hora actual és menor a les 06 : 00 : 00.

En el primer cas es fa un `count` a la taula `table_plc` de la base de dades de tots aquells registres amb data d'avui i hora compresa entre l'hora inicial i final calculades amb els mètodes comentats a dalt. A més, s'afegeixen com a condicions: `Error=1` i `Break = false`.

En el segon cas la consulta és més simple, ja que només comprova si l'hora actual és més gran que l'hora d'inici del torn per al dia d'avui. Les condicions anteriors d'*error* i *break* es mantenen.

Per últim, si cap de les condicions anteriors s'han complert significa que el torn de matí encara no ha començat. Això genera un problema, perquè significa que el torn de nit ja s'ha estès durant dos dies diferents. Llavors, la consulta que s'ha de realitzar ha de comprovar tant els registres escrits a partir de les 22:00:00 del dia anterior com els registres escrits fins l'hora actual. Això s'aconsegueix mitjançant la funció `timedelta()` de la biblioteca `datetime` de Python.

Per acabar, si cap de les condicions anteriors s'han complert, vol dir que el torn de matí encara no ha començat. Això genera un problema, perquè significa que el torn de nit ja s'ha estès durant dos dies diferents. Llavors, la consulta que s'ha de realitzar ha de comprovar tant els registres escrits a partir de les 22 : 00 : 00 del dia anterior com els registres escrits fins a l'hora actual. Això s'aconsegueix mitjançant la funció `timedelta()` de la biblioteca `datetime` de Python.

El codi que s'aconsegueix finalment és el següent. `current_time` és una variable local que crida la funció `strftime` del mòdul `datetime`. Per veure més informació d'aquest mòdul es pot llegir la documentació oficial [Fou23d]:

```
def get_num_error(self):
    current_time = datetime.now().strftime("%H:%M:%S")

    if ('06:00:00' <= current_time < '22:00:00'):
            sql_query = "SELECT COUNT(*) FROM table_plc WHERE Date = '{}' AND
                Hour >= '{}' AND Hour < '{}' AND Error = 1 AND Break =
                FALSE".format(time.strftime("%D"), self.start_shift_time,
                self.end_shift_time)

    elif (current_time >= "22:00:00"):
        sql_query = "SELECT COUNT(*) FROM table_plc WHERE (Date = '{}' AND Hour
            >= '{}') AND Error = 1 AND Break =
            FALSE".format(time.strftime("%D"), self.start_shift_time)
```

```
        else:
            sql_query = "SELECT COUNT(*) FROM table_plc WHERE ((Date = '{}' AND
                Hour >= '{}') OR (Date = '{}' AND Hour < '{}')) AND Error = 1 AND
                Break = FALSE".format((datetime.today() - timedelta(days =
                1)).strftime("%D"), self.start_shift_time, time.strftime("%D"),
                self.end_shift_time)

        return self.object_db.write_to_db(self.sql_connection, self.sql_cursor,
            sql_query)
```

D'altra banda, sempre que hi ha un temps de descans o de manteniment planificat es crida la funció `update_break_true(self)` que actualitza l'últim registre de la taula `table_plc` de la base de dades posant `Break=True`. La consulta que s'executa és la següent

```
UPDATE table_plc SET Break = True WHERE id = (SELECT id FROM table_plc ORDER BY
    id DESC LIMIT 1)
```

Productivitat

Per al càlcul de la productivitat necessitem els valors t_w, t_b i t_e representats en les variables d'instància `current_work_time`, `current_planned_stop_time` i `current_error_time`. Aquests valors ja s'han calculat prèviament en el càlcul de la disponibilitat. També necessitem saber el total de les peces fabricades fins a l'hora actual. Aquest valor s'obté fent una consulta a la base de dades a través del mètode `total_pieces_fabricated(self)`.

Aquest mètode fa un sumatori dels registres `Ok` i `Nok` de la taula `table_plc` de la base de dades. D'igual forma que en les funcions anteriors, hem de tractar com correspon els horaris definits. Per fer-ho se segueix la mateixa estructura comentada en la subsecció 6.2.2, modificant la consulta de la següent forma:

```
SELECT SUM(OK + NOK) FROM table_plc WHERE condició
```

On `condició` correspon a les condicions definides dins l'estructura de control de flux que ja s'ha comentat anteriorment.

Qualitat

Per trobar la qualitat hem de saber tant el total de peces produïdes — tan bones com dolentes — com el total de peces bones. El total de peces fabricades comptant les bones i les dolentes s'obté a través del mètode `total_pieces_fabricated(self)` definit en la subsecció 6.2.2.

El total de peces bones o *ok* s'obté a partir d'una funció molt similar, `total_pieces_ok_fabricated(self)`, però on canvi la consulta canvia lleugerament. Ara el sumatori no és `SUM(OK + NOK)` sinó `SUM(OK)`.

6.2.3. Mètodes de càlcul

Permeten fer els càlculs de l'OEE i els seus paràmetres. Són un total de vuit mètodes, els quals calculen l'OEE, la disponibilitat, la productivitat i la qualitat, així com totes les dades

necessàries per a trobar-los.

```
def get_oee(self)
def get_availability(self)
def work_time(self)
def planned_stops_time(self)
def update_break_true(self)
def error_time(self)
def get_performance(self)
def get_quality(self)
```

Disponibilitat

La disponibilitat es calcula mitjançant tres valors que s'han d'anar actualitzant a mesura que avança el temps: el temps de treball total t_w, el temps dels descansos planificats t_b i el temps perdut a causa dels errors i avaries t_e. Les variables d'instància `current_work_time`, `current_planned_stop_time` i `current_error_time` corresponen a aquests valors.

`Current_work_time` és ben fàcil de calcular. Primer, la funció `work_time(self)` troba l'hora actual. Després, resta l'hora inicial del torn de l'hora actual. L'atribut `seconds` de la classe `datetime.timedelta` inclosa dins del mòdul `datetime` retorna en segons el resultat, ja que totes les operacions que realitzen càlculs de temps utilitzen els segons com a unitat de mesura. Finalment el codi resultant és:

```
def work_time(self):
    current_time = datetime.now().strftime("%H:%M:%S")
    self.current_work_time = (datetime.strptime(current_time, "%H:%M%S") -
        datetime.strptime(self.start_shift_time, "%H:%M%S")).seconds
    return self.current_work_time
```

La funció `planned_stops_time(self)` té com a objectiu calcular el temps previst de les parades i emmagatzemar-lo a la variable d'instància `current_planned_stop_time`. Primerament, s'obté l'hora actual i posteriorment es realitzen els càlculs corresponents seguint els horaris definits a la taula 6.3. És important notar que es descarten les últimes hores del torn de nit, ja que segons el control de flux definit aquestes hores corresponen al temps de descans i la disponibilitat seria zero, cosa que no és correcta. En aquesta part del càlcul, s'han de distingir quatre possibles casos:

- Si un manteniment planificat coincideix amb un descans.
- En cas que no coincideixin el manteniment previst i la parada planificada, o bé si algun d'ells no està definit.
- Si en el torn no hi ha ni manteniments ni parades previstes.

En el primer cas, s'incrementa `current_planned_stop_time` el temps definit en la variable `interval_duration`. Després, s'actualitza el registre `Break` de la taula `table_plc` perquè valgui `True` i es resta una unitat tant a `maintenance_duration` com a `break_duration`. És rellevant notar que el càlcul es deixa de realitzar en el moment que aquestes variables siguin menors que zero — inicialment estan inicialitzades a -1.

En el segon cas es fa el mateix: s'incrementa `current_planned_stop_time`, s'actualitza la base de dades i es descompta la variable que pertoqui. Això no obstant, es fa separadament per distingir les parades planificades dels manteniments. En l'últim cas no es realitza cap càlcul.

Tenint tot això en compte la funció queda de la següent manera:

```
def planned_stops_time(self):
    current_time = datetime.now().strftime("%H:%M:%S"

    # TODO - Configure according to the requirements and the shifts of the
        client.
    if (current_time < "22:00.00")

        # We consider whether the planned stops coincide in time with the
            planned maitenance.
        if ((self.break_time != -1) and (current_time >= self.break_time) and
            (self.break_duration > -1) and (self.maintenance_time != -1) and
            (current_time >= self.maintenance_time) and
            (self.maintenance_duration > -1)):
            self.update_break_true()
            self.current_planned_stop_time += self.interval_duration;
            self.maintenance_duration -= 1
            self.break_duration -=

        elif (self.break_time != -1 and current_time >= self.break_time and
            self.break_duration > -1):
            self.update_break_true()
            self.current_planned_stop_time += self.interval_duration;
            self.break_duration -=

        elif (self.maintenance_time != -1 and current_time >=
            self.maintenance_time and self.maintenance_duration > -1):
            self.update_break_true()
            self.current_planned_stop_time += self.interval_duration;
            self.maintenance_duration -=

    return self.current_planned_stop_time
```

El temps perdut a causa dels errors i avaries s'obté mitjançant la funció `error_time(self)`. Aquesta funció multiplica el nombre d'errors trobat en el mètode `get_num_error(self)` per la variable d'instància `interval_duration`. És important la següent condició:

```
Error = 1 AND Break = FALSE
```

Amb això s'aconsegueix ignorar els errors que puguin haver ocorregut durant una parada o un manteniment planificats. Si no es fa això, l'error se sumaria sempre i podria fer que la disponibilitat tingués valors negatius. Per exemple, si en el torn anterior s'hagués produït un error encara no solucionat — disponibilitat seria 0 —, i en el torn actual ja s'estigués en temps de descans, el resultat obtingut seria negatiu, ja que es restaria al temps treballat el temps d'error, que en aquest cas seria el mateix que el temps treballat, més el temps de descans. La funció resultant és ben senzilla:

```
def error_time(self):
    self.current_error_time = self.get_num_error() * self.interval_duration
```

```
    return self.current_error_time
```

La funció `get_availability(self)` calcula la disponibilitat segons els passos descrits en el capítol 3, i arrodoneix el resultat mitjançant la funció `round` a dos decimals. Si el temps de les parades planificades i dels manteniments programats és el mateix que el temps total que ha passat dençà que ha començat el torn, genera l'excepció `ZeroDivisionError` i fica la disponibilitat al 0%. Finalment, retorna la variable d'instància `availability`.

Productivitat

Respecte del procés de calcular la productivitat, aquest càlcul és senzill i permet fer-lo usant només dues funcions. Ja s'ha explicat en la subsecció 6.2.2 com s'obtenen la suma de totes les peces fabricades fins a l'instant actual, així que només queda explicar la funció que calcula directament aquest paràmetre.

`Get_performance(self)` obté el total de les peces fabricades, tant les bones «*ok*» com les dolentes «*nok*». Després divideix el resultat aconseguit per t_t, el qual s'obté directament. Si aquest és 0 crida l'excepció *ZeroDivisionError* i posa la productivitat al 0%. Si l'excepció és qualsevol altra, imprimeix un missatge d'error i fa el mateix amb la productivitat. Es pot trobar més informació sobre els errors típics de Python i com treballar amb excepcions a la documentació oficial [Fou23f]. A la mateixa documentació, també hi ha un apartat dedicat a explicar el funcionament d'aquesta característica i les diferents excepcions que s'hi inclouen [Fou23b].

Així mateix, és important notar que acabem comprovant que el resultat obtingut no sigui negatiu o superior al 100%. Això és degut al fet que en les proves realitzades s'ha observat que el PLC ha *fabricat* algunes peces de més, de forma que hi havia registres que en lloc de tindre sis peces en total en tenien set. Com que el cicle de producció és de 10,00 s per peça, això a vegades ha provocat que la productivitat fos major al 100%, cosa que no hauria de passar mai.

Així mateix, és important destacar que és necessari comprovar que el resultat obtingut no sigui superior al 100% ni negatiu. Aquesta mesura es deu al fet que durant les proves realitzades s'ha observat que el controlador PLC ha produït algunes peces de més, de manera que hi havia registres que contenien set peces en lloc de les sis esperades, incloent-hi les peces correctes i les defectuoses. Això, justament pel fet que el cicle de producció és de 10,00 s per peça, ha provocat que en aquests casos concrets la productivitat superi el 100%, cosa que no tindria de passar mai.

Tenint tot això present, la funció resultant és la que segueix. És important notar que el resultat s'arrodoneix a dos decimals mitjançant la funció `round` [Fou23c].

```
def get_performance(self):
    try:
        self.performance = round((100 * self.total_pieces_fabricated() /
            ((self.current_work_time - self.current_planned_stop_time -
            self.current_error_time) / self.cycle_time)), 2)

    except ZeroDivisionError:
        self.performance = 0

    except Exception as e:
        print('Error in "get_performance". Error: ' + e)
        self.performance = 0
```

```
        if (self.performance < 0): self.performance = 0
        if (self.performance > 100): self.performance = 100

        return self.performance
```

Qualitat

Com ja s'ha comentat, la qualitat és el resultat de dividir el total de les peces fabricades correctament pel total de peces fabricades, incloent-hi les defectuoses. Així, la funció `get_quality(self)` retorna el resultat d'aquesta operació arrodonit a dos decimals. Igualment, gestionem les excepcions per si encara no s'han fabricat peces, ja que del contrari retornaria un resultat indefinit [Bri06].

Eficiència general dels equips (oee)

Només hi ha una funció que calcula l'OEE. `Get_oee(self)` fa el producte descrit en la secció 3.1.4. Aquesta funció retorna la variable d'instància `oee` amb el resultat arrodonit a dos decimals.

7. Interfície web i servidor

Per acomplir aquest projecte i poder visualitzar les dades de la producció i l'OEE calculats s'ha creat un servidor Flask. Aquest servidor ha de permetre visualitzar una pàgina web simple i adaptativa on es mostrin gràfiques i dades a partir de la base de dades explicada en el capítol 4. D'aquesta forma la Raspberry PI ha de contenir tant la base de dades, com el programa de comunicació amb el PLC descrit en el capítol 5 com el servidor Flask amb la seva pàgina web.

Amb tot això, aquest capítol s'estructura a través de dos punts diferents:

- La creació de la pàgina web i del servidor Flask.
- La instal·lació de l'entorn Linux i configuració de la Raspberry PI.

7.1. Flask i aplicació web

Flask és un entorn de treball en miniatura que permet la creació d'aplicacions web de forma senzilla i eficient [Ron22]. Gràcies a la seva estructura lleugera i flexible és una eina molt popular perquè permet als desenvolupadors adaptar-lo a les seves necessitats. A diferència d'altres entorns de treball, Flask no té cap capa d'abstracció o funcionalitat incorporada, el que es tradueix en el fet que els desenvolupadors poden implementar només aquelles funcionalitats que necessiten.

Per altra banda, en el seu dia es va decidir que l'aplicació web havia de complir les següents característiques [Ing]:

- Sistema d'usuaris per controlar l'accés a l'aplicació.
- Calendari per marcar els dies festius de l'any.
- En la pàgina principal s'ha de poder escollir entre les «n» màquines de la línia, tot i que per culpa de què només es disposa d'un banc de proves només s'hi ha inclòs una.
- Així mateix, la pàgina principal ha d'incloure les gràfiques dels paràmetres calculats, de l'OEE així com les peces fabricades i la distribució del temps. També s'han de poder visualitzar cada un dels paràmetres que composen l'índex: la disponibilitat, la productivitat i la qualitat.
- Gràfiques que representen els mateixos paràmetres per a un torn determinat.
- Connectivitat amb el PLC directament a través de TCP/IP.
- Connectivitat a Internet a través d'Ethernet. Per culpa de limitacions tècniques, la connectivitat a Internet s'ha dut a terme mitjançant wifi.
- Disseny web adaptatiu.

Tenint tot això present, s'ha volgut fer una web tan senzilla com sigui possible. S'ha triat el model SPA per al seu disseny i implementació. SPA és l'acrònim de *Single-page application*, i és una tècnica de disseny web on tota la informació és mostra en una única pàgina [Fla20]. Amb aquesta metodologia s'intenta proporcionar una experiència d'usuari fluida i ràpida mitjançant la descàrrega de tot el contingut una sola vegada, enfront dels dissenys tradicionals on el contingut es va descarregant a mesura que l'usuari avança per les diferents pàgines. Així, en els dissenys SPA el contingut es carrega dinàmicament a mesura que es va requerint, ja sigui a través de les accions dels usuaris o bé mitjançant codi inserit en els mateixos recursos. Per exemple, en l'aplicació dissenyada tot el contingut HTML i JavaScript es carrega una sola vegada a l'inici, però mitjançant AJAX — acrònim per a *Asynchronous JavaScript and XML* — es va actualitzant cada cert temps. AJAX és una tècnica de desenvolupament web per aconseguir crear pàgines web asíncrones. D'aquesta forma el contingut web s'executa en el client al mateix temps que per sota es manté la comunicació asíncrona amb el servidor. Per trobar més informació sobre AJAX es pot consultar la web de W3Schools [W3S23].

En l'apèndix C es poden veure captures de pantalla de la interfície d'usuari de l'aplicació final.

7.1.1. Estructura

Com es pot observar en la imatge 7.1, una aplicació Flask típicament s'estructura de la següent forma:

- `app` o carpeta arrel, conté tots els codis `.py` i de configuració del servidor. Sol tindre un nom semblant al nom de l'aplicació creada.
- `templates` conté els codis HTML.
- `static` guarda fitxers estàtics i immutables, com estils CSS, JavaScript, imatges, documents, etc. Aquests se solen emmagatzemar en subcarpetes segons el seu tipus.

Així, i com que l'aplicació resultant es basa en l'estil SPA descrit abans, només s'han creat un fitxer HTML i un arxiu de configuració `.py`. Per altra banda, el directori `static` conté subdirectoris per guardar la base de dades de prova, el fitxer d'estil CSS, imatges i codi JavaScript.

7.1.2. Configuració inicial

A causa del fet que en un inici no es disposava d'una Raspberry PI ni d'un banc de proves habilitat s'ha començat el projecte per la creació del servidor web i l'aplicació Flask. D'aquesta forma, per fer el disseny inicial i les configuracions prèvies s'ha utilitzat *Windows Subsytem for Linux* o WSL pel seu acrònim [Mic22b]. Aquesta és una capa de compatibilitat pròpia dels sistemes operatius de Microsoft, la qual permet executar programes i fitxers basats en Linux sobre un sistema operatiu Windows gràcies a una interfície que simula un nucli de Linux. Tot el procés es du a terme en un entorn que no conté codi Linux i que es totalment privatiu [Ger16].

S'ha iniciat la preparació de l'entorn per treballar localment en la creació del servidor. Això implica la instal·lació i configuració de totes les eines i *plugins* necessaris. Per a tal fi, s'ha creat un fitxer `requeriments.txt` amb tots els paquets necessaris que pot ser instal·lat mitjançant

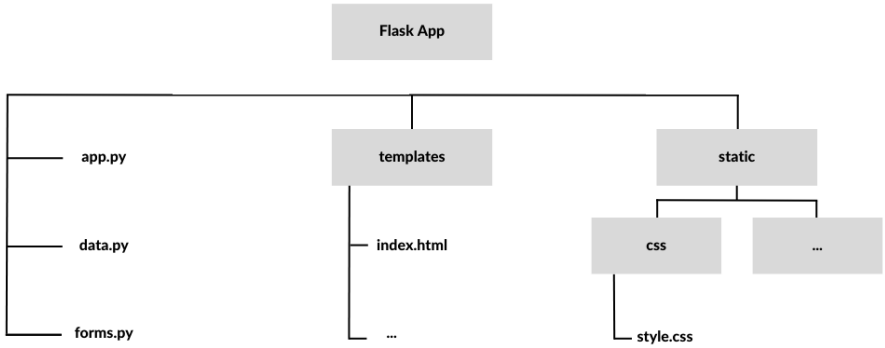

Figura 7.1.: Estructura típica d'una aplicació Flask. Diagrama basat en la secció *Folder structure for a Flask app* de `la web oficial del projecte`.

«pip». «Pip» és un sistema de gestió de paquets que permet la instal·lació, manteniment i administració d'aquests d'una forma senzilla i intuïtiva [dev23].

```
$ pip install -r requeriments.txt
```

A continuació s'explica per a què serveixen alguns d'aquests paquets. Si es vol saber més informació sobre els paquets disponibles per a aquest llenguatge, és possible consultar l'Índex de paquets de Python a [Fou23e].

Flask-Bootstrap

`Flask-Bootstrap` és una extensió que permet integrar Bootstrap amb Flask. Bootstrap és un conjunt d'eines de codi obert que facilita el disseny de pàgines web de forma senzilla i elegant, així com una àmplia biblioteca de recursos per als desenvolupadors «frontals». Aquesta extensió proporciona plantilles predefinides de Jinja2 amb elements de Bootstrap ja integrats, el que facilita la creació de llocs web moderns, sofisticats i adaptables.

Flask-login

Permet la gestió, autenticació i creació d'usuaris per aquelles aplicacions web que ho requereixin. Disposa funcions, decoradors i eines que permeten gestionar tot el procés d'autenticació d'usuaris, inicis de sessions, proteccions de rutes, validacions, etc. Amb aquest paquet també és possible personalitzar el comportament de l'autenticació per adaptar-se a la idiosincràsia de cada aplicació.

En concret, aquesta extensió permet:

- Guardar l'identificador de l'usuari durant la sessió.
- Permetre fer inicis i finals de sessió.
- Permetre l'accés a certes vistes únicament als usuaris que s'hagin autentificat.
- Mantenir les sessions obertes inclús després que l'usuari tanqui el navegador.
- Protecció de les galetes de la sessió enfront tercers.

Flask-SQLAlchemy, SQLAlchemy

Aquesta extensió permet treballar amb bases de dades SQL de forma senzilla i eficient. Proporciona una capa d'abstracció sobre la base de dades, amb la que es permet de forma transparent crear models d'objectes i manipular-los mitjançant consultes. A més simplifica el desenvolupament permetent manipular objectes de la base de dades com objectes Python convencionals gràcies al fet que integra *Object Relational Mapping* o ORM per les seves sigles en anglès.

Flask-WTF, WTForms

`WTForms` és una biblioteca de Python que permet crear formularis web fàcilment. Proporciona diversos tipus de validacions, així com la generació de camps de formularis personalitzats. Aquesta biblioteca pot ser integrada en aplicacions Flask mitjançant l'extensió `Flask-WTF`. Aquesta extensió proporciona protecció d'enfront atacs CSRF — en anglès *Cross-Site Reques Forgery* — en formularis web, així com funcionalitats per a la seva validació.

Jinja2

Per la creació de llocs web dinàmics escrits en Python es pot fer servir el motor Jinja2. Jinja2 és un llenguatge basat en text, per la qual cosa es pot fer servir per generar qualsevol marcatge o codi font.

`Flask-Jinja2` és una extensió que proporciona integració amb aquest motor. Amb el seu ús, es poden configurar diverses opcions de Jinja2 en una aplicació Flask, com ara directoris de plantilles o sintaxi personalitzada, així com es permet millorar la gestió de plantilles en l'aplicació — molt útil per futures ampliacions.

WSGI

Web Server Gateway Interface o WSGI pel seu acrònim és una especificació que descriu la comunicació entre un servidor web i una aplicació escrita amb Python-Flask, així com la forma en què es poden arribar a encadenar diverses aplicacions. D'aquesta forma, WSGI permet la comunicació entre el servidor i l'aplicació web actuant d'intermediari: el servidor espera peticions des de l'exterior, mentre l'aplicació fa una petició i espera que el servidor les hi passi per tal de tractar-les correctament i donar una resposta. La versió actual d'aquesta especificació és la 1.0 segons *Python Enhancement Proposals* 333 [Fou22].

Mitjançant els paquets `werkzeug` `pyuwsgi` — es necessari tindre el servidor Apache instal·lat — es proporcionen unes funcions per gestionar tasques comuns en el desenvolupament web. Algunes de les tasques que es poden gestionar amb aquest paquet són: analitzadors d'URLs, sol·licituds d'usuaris, respostes HTTP, entre d'altres.

Protecció CSRF

El CSRF o *Cross-site request forgery* és un *exploit* maliciós d'un lloc web en el que comandes no autoritzades són transmeses per un usuari en el qual el lloc web confia. D'aquesta forma, un atac CSRF força el navegador web de la víctima a enviar una petició a una aplicació web vulnerable, la qual realitza l'acció elegida a través de la víctima. Un exemple molt clar del funcionament d'aquest «explotador» es recull en [Mic22a].

En l'aplicació creada, la forma d'evitar aquest explotador ha estat la creació d'un testimoni únic compartit només entre el servidor i el client. D'aquesta forma, cada cop que s'envia una

sol·licitud sensible el servidor espera que contingui la clau CSRF assignada. Si aquesta clau és incorrecta, el servidor ha de rebutjar la petició. Com és obvi, per seguretat aquests testimonis no es guarden mai en les galetes del client.

Werkzeug.security

Com s'explica en 7.1.2, `werkzeug` és un conjunt d'eines per treballar amb WSGI. D'aquesta forma permet utilitzar-lo com la biblioteca adjacent perquè encapsula moltes funcionalitats típiques d'un entorn web. En concret, una de les funcionalitats que proporciona és el xifratge de contrasenyes mitjançant el mòdul `werkzeug.security`.

Amb això, podem fer servir les següents funcions per encriptar o desencriptar la informació dels formularis: `generate_psw_hash()` i `check_psw_hash()`.

Amb la primera generem una empremta electrònica que ha de permetre ocultar la contrasenya o altres textos que necessitem amagar. Aquesta funció accepta fins a tres paràmetres: el text de la contrasenya que es vol ocultar, el mètode d'encriptat i la longitud del *salt length*, que és un valor aleatori que s'afegeix a la contrasenya per evitar que dos usuaris amb la mateixa paraula clau tinguin la mateixa empremta.

Amb la segona validem la contrasenya introduïda a través del formulari d'inici de sessió amb l'empremta electrònica guardada en la base de dades. Accepta dos paràmetres, l'«empremta electrònica» i el text a comparar, i retorna un valor booleà.

7.1.3. Codi Python

Aquesta aplicació la conformen tres fitxers diferents de codi Python. D'aquesta forma, `index.py` conté totes les rutes i configuracions principals de l'aplicació; `forms.py` conté les validacions dels formularis i `models.py` les crides a la base de dades i la gestió dels usuaris.

`api.py`

El codi Python del fitxer `api.py` és bastant simple, ja que com s'està usant un model SPA només ha de contenir una pàgina. Així, el codi Python de la pàgina principal «`def index()`» només ha de cridar la funció `render_template("index.html")` que retorna el codi HTML d'aquesta, contingut en el directori `templates`.

Així mateix, i com s'explica posteriorment en la subsecció 7.1.5, per visualitzar les dades es creen rutes diferents. D'aquesta forma, per cada petició a la base de dades o gràfica a visualitzar es crea una ruta diferent. En Flask, els *endpoints* són funcions que s'executen un cop el servidor rep una sol·licitud a un URL concreta. Per crear aquestes direccions, es fa servir una sintaxi específica:

```
@app.route("/", methods = ["GET", "POST"]
def index():
    """
    Es realitzen les accions que requereixi la pàgina.
    """
    # Code according the app functionalities.
    # Call HTML template.
    return render_template("index.html")
```

Per cada ruta de l'aplicació es crea una funció que conté les accions que aquesta ha de realitzar. Un cop aquestes han acabat es crida el codi HTML corresponent emmagatzemat dins la ruta `templates` mitjançant la funció `render_template()`, que accepta com a paràmetre el nom del fitxer HTML. Si la pàgina no ha de realitzar cap acció — validacions de formularis, consultes a bases de dades, etc. — es crida directament. Així mateix, aquesta funció també accepta paràmetres que posteriorment poden ser llegits i tractats en el codi HTML. Mitjançant els decoradors `@app.route("nom_de_la_ruta", methods = ["GET", "POST"]` s'informa el navegador que mostri la pàgina web `index.html` quan l'usuari afegeix la ruta `/` en l'URI, així com el mètode de la petició que ha de fer. Per altra banda, es pot indicar que la mateixa pàgina sigui accessible des de dues rutes diferents afegint els decoradors necessaris a la funció.

El control d'usuaris es realitza mitjançant un formulari i Flask-Login. Aquest formulari crida la ruta `login` mitjançant el mètode `post`. Un cop s'ha validat l'accés, el formulari es deixa de mostrar i en el seu lloc es mostra un botó de *logout*, el qual al seu torn crida la ruta `logout` que tanca la sessió de l'usuari. Si l'usuari ha iniciat la sessió correctament, veurà noves vistes a la pàgina principal que li permetran configurar i accedir a altres dades de la base de dades.

Per actualitzar els gràfics també s'han creat rutes diferents. D'aquesta forma, quan es vol actualitzar-ne un es crida la ruta corresponent. En el codi associat a aquesta ruta es realitzen les accions necessàries per obtenir les dades a visualitzar, les quals normalment són consultes a la base de dades.

`forms.py`

El mòdul `forms.py` conté una classe `LoginForm(Form)`. Aquesta classe hereta de la classe `Form` de Flask, per la qual cosa es fa servir per a definir un formulari d'inici de sessió. Aquesta classe defineix tres camps diferents:

- `email` és un camp d'entrada de text, on els usuaris han d'ingressar el seu identificador.
- `psw` és un camp d'entrada de contrasenya. Això vol dir que els caràcters introduïts queden amagats a l'usuari.
- `remember_login`, un camp de verificació booleà. Amb això es permet recordar l'inici de sessió.
- `submit`, un botó que permet enviar les dades del formulari.

Així mateix, tant `email` com `psw` tenen la validació `DataRequired()`. Amb això s'obliga a haver d'introduir tots dos camps correctament per poder realitzar l'inici de sessió. Quan es pitja el botó `submit` el formulari s'envia al servidor perquè aquest pugui processar la sol·licitud d'inici de sessió.

`models.py`

Aquest mòdul conté les classes i funcions necessàries per gestionar els usuaris, el xifratge de les contrasenyes i les consultes a la base de dades. D'aquesta forma, `models.py` té les classes següents:

- `User` conté mètodes per a xifrar la contrasenya de l'usuari.
- `Plc` fa les consultes a la taula `table_plc` de la base de dades.

La classe `User` conté mètodes per xifrar la contrasenya de l'usuari. D'aquesta forma, `self_pws(self,pasw)` crida el mètode `generate_password_hash(psw)` per encriptar el text passat com a paràmetre. Per la seva banda, `check_psw(self,psw)` crida la funció `check_psw_hash(self,psw)` per validar la contrasenya enviada mitjançant el formulari d'inici de sessió amb l'empremta digital guardada a la base de dades.

D'altra banda, la classe `Plc` fa les crides corresponents a la base de dades per tal de poder visualitzar els gràfics correctament actualitzats. També, i si fa falta en un futur, ha de poder permetre fer canvis a la taula `table_shfits`. De moment només s'han programat funcions amb la finalitat que facin consultes a la base de dades. Per facilitar el pas dels resultats cap al codi JavaScript que gestiona i configura les gràfiques i tota la informació que s'ha de veure, s'ha fet una crida per cada gràfica. Aquesta consulta tots els paràmetres necessaris per al gràfic en qüestió i els afegeix a una llista, que serà la que retorni finalment el mètode.

7.1.4. HTML i CSS

Per al disseny web s'han fet servir les tipografies de Google Fonts. En concret, s'ha triat la tipografia Rubik per tot el disseny, tant dels títols com dels cosos del text. D'altra banda, per al disseny frontal s'ha emprat Bootstrap. Bootstrap és una biblioteca o conjunt d'eines disponibles en diverses plataformes que permet la creació d'aplicacions web elegants i adaptables de forma senzilla. Gràcies al fet que conté plantilles, formularis, botons, requadres, menús, colors, etc. és molt senzill per als programadors crear dissenys web moderns, elegants i fàcils d'utilitzar. A diferència d'altres entorns de treball, Bootstrap només proporciona eines per al treball sobre l'aplicació frontal i no de fons com sí que fan altres biblioteques. Es pot trobar més informació sobre aquest entorn en la web oficial del projecte [Tea23].

En el disseny s'han fet servir diversos elements d'aquest entorn. Entre ells, un dels més importants, ja que conté el formulari d'inici de sessió i altra informació rellevant, és la barra lateral que s'estén al llarg de tota la banda esquerra de la web. En aquesta barra es mostra el logotip i informació de l'empresa, els últims valors de l'OEE, l'hora en temps real, i el formulari d'inici de sessió o botó de fi de sessió segons convingui. En concret, s'ha agafat un model d'exemple de la pàgina [Tea], el qual s'ha modificat per encaixar en el disseny de l'aplicació.

Per tal de fer la pàgina web responsiva, s'han creat diverses regles CSS que permeten readaptar i inclús amagar alguns elements segons la resolució de la pàgina. La forma de crear aquestes regles és mitjançant `@media`. Aquesta regla permet definir diversos estils CSS per diferents mitjans o resolucions, i la seva sintaxi és:

```
@media screen and (max-width: 1444px) {
  #sidebar-container{
    display: none;
  }
  .graphBox {
    grid-template-columns: 1fr 1fr 1fr;
    vertical-align: middle;
    height: auto;
  }
}
```

On `screen` — també pot ser `print` — indica que el dispositiu en els que s'ha d'aplicar és en pantalles. `Max-width:1444px` indica que la resolució de les pantalles on s'aplicarà aquest codi han de tindre una amplada màxima de 1444,00 px. Entre les claus s'indiquen les regles que s'han d'aplicar en aquesta resolució.

Un altre punt important del codi HTML és la forma d'ocultar o mostrar informació segons si un usuari ha estat validat correctament o no. Això es fa mitjançant plantilles de Jinja2. Jinja2 és un motor de plantilles per a Python configurat per defecte en Flask, de forma que facilita la separació entre l'escriptura de vistes, la lògica i la presentació. En el punt 7.1.3 ja s'ha mostrat un exemple del funcionament de Jinja2 mitjançant la funció `render_template`. La forma de funcionar de Jinja2 és senzilla i permet el pas d'informació i de dades de forma ràpida i eficaç. Amb tot això, aquest punt s'ha realitzat mitjançant una sentència condicional com la següent:

```
{% if 'username' not in session %}
    <!— Contingut HTML de la Vista 1. —>
{% else %}
    <!— Contingut HTML de la Vista 2. —>
{% endif %}
```

És important notar que la sentència `if/else` està correctament escrita en Python, però que tant a l'inici com al final s'han afegit els caràcters «`{%`» i «`%\}`» respectivament. Això es fa per marcar que el codi comprès entre aquests caràcters és codi Python i correspon a una sentència condicional. Per altra banda, una forma de passar paràmetres — per exemple valors llegits de la base de dades — és fer-ho escrivint el nom de la variable del codi Python contingut dins la funció del mòdul `api.py` entre els caràcters

```
<h4 style="margin-bottom: 1rem; text-align: left;">
    Welcome {{ username }}
    </h4>
```

Per altra banda, el contingut JavaScript que permet crear els gràfics interactius — entre altres coses — s'ha afegit a la pàgina mitjançant l'etiqueta `<script>`. A continuació es mostra un exemple real on el directori on s'ubica el codi s'indica mitjançant l'atribut `src` de l'etiqueta.

```
<script src = "static/sj/clock.js"></script>
```

7.1.5. JavaScript

Per al disseny dels gràfics i d'altres recursos, per exemple el rellotge i la data que apareixen en la barra lateral, s'ha fet servir JavaScript. Com que el contingut s'ha d'actualitzar de forma asíncrona a la resta de la pàgina, s'ha usat AJAX per a tal fi. AJAX (*Asynchronous JavaScript and XML* és un grup de tecnologies usades per desenvolupar aplicacions web que requereixen carregar continguts de forma dinàmica i periòdica. D'aquesta forma els petits paquets de dades es bescanvien amb el servidor, per la qual cosa les pàgines web no s'han de carregar cada cop que es requereixi actualitzar la informació o que l'usuari faci una acció. Això permet que l'usuari pugui interactuar amb la pàgina sense patir les molestes interrupcions que implica haver de recarregar-la de nou, de forma que la interacció amb el servidor per part del client només es du en aquelles parts específiques que ho requereixen. Per fer un resum, AJAX es basa en les següents tecnologies [Cor21]:

- XHTML i CSS per presentar la informació.

- DOM — *Document Object Model* —, per visualitzar o interactuar de forma dinàmica amb el contingut.
- XMLHTTPRequestper és un objecte per manipular les dades de forma asíncrona amb el servidor.
- XML, HTML I XSLT per l'intercanvi i manipulació de dades.
- JavaScript per enllaçar sol·licituds o informació.

En aquesta aplicació s'ha fet servir AJAX per actualitzar els gràfics. Aquests, al seu torn, s'han creat mitjançant Chart.js. Chart.js és una biblioteca pròpia de JavaScript que permet la creació i visualització de gràfics interactius. Mitjançant HTML5 i l'element `<canvas>` permet dibuixar gràfics adaptables fàcilment. Aquesta biblioteca permet canviar els colors dels gràfics mostrats, així com el seu tipus, ja que pot dibuixar gràfics d'àrea, de barres, de bombolles, circulars, en forma de dònut, d'àrea polar, d'aranya o de dispersió. També permet crear gràfics mixOs combinant dades i estils diferents. En la seva pàgina web `https://www.chartjs.org/docs/latest/` es pot trobar tota la documentació referent a aquesta llibreria.

Per al disseny de l'aplicació s'han triat dos tipus de gràfics diferents: gràfics en forma de dònut i gràfics de barres. El primer és com el circular però amb un forat al mig, i facilita la visualització de múltiples dades en un sol gràfic. El segon s'utilitza per mostrar els paràmetres de l'OEE dels torns anteriors per separat, ja que era un requisit que provenia de l'inici del projecte. Aquest es mostra horitzontalment.

Chart.js es pot instal·lar baixant el repositori de GitHub o fent servir els contenidors de programari *cdnjs* o *JSDelivr*. Tanmateix, la forma més fàcil d'instal·lar-lo és utilitzar el sistema de gestió de paquets de Node.js `npm`:

```
npm install chart.js
```

Abans de poder crear gràfics s'ha d'importar la llibreria al codi HTML. Per fer-ho s'usa l'etiqueta `<script>` com es descriu en la seva pàgina. És important notar que per incloure els gràfics en la interfície d'usuari s'ha de fer servir l'element `<canvas>` i assignar un identificador diferent a cada gràfic.

Per crear un gràfic amb aquesta llibreria es fa servir el següent codi, canviant els paràmetres necessaris en cada cas. Aquests són gràfics altament «personalitzables» on la sintaxi utilitzada per crear-los correspon al format JSON. Un exemple típic és el següent.

```
 const ctx = document.getElementById('myChart');
new Chart(ctx, {
    type: 'bar',
    data: {
      labels: ['Red', 'Blue', 'Yellow', 'Green', 'Purple', 'Orange'],
      datasets: [{
        label: '# of Votes',
        data: [12, 19, 3, 5, 2, 3],
        borderWidth: 1
      }]
    },
    options: {
      scales: {
        y: {
```

```
            beginAtZero: true
          }
        }
      }
});
```

Aquest codi crea un gràfic de barres. Com que s'han fet servir gràfics en forma de dònut i de barra, podem usar tant `bar` com `doughnut` dins l'opció `type`. Dins de `labels` es pot incloure el nom de les etiquetes — valors — que es visualitzarà, mentre que `label` indicia el títol d'aquest i `data` conté una llista amb els valors que s'assignen. Per actualitzar els gràfics amb els valors llegits des de la base de dades s'ha fet servir una funció que crida una de les rutes definides en `api.py`. Un exemple de com es realitza aquesta acció és el següent (en l'exemple el gràfic només actualitza un valor).

```
function updateChart() {
    $.get("/route_defined_in_apipy", function(data) {
        chart.data.datasets[0].data[0] = data.datareaded;
        chart.update();
    })
}
```

En aquest exemple es defineix una funció `updateChart()` que fa servir jQuery per enviar una sol·licitud `get` a una ruta específica definida dins del mòdul `api.py`. Un cop s'ha rebut resposta s'actualitzen les dades del gràfic existent. La funció `.get{}` de jQuery té dos arguments: el primer és la ruta a la qual s'envia la sol·litud, mentre que el segon és una funció que s'executarà quan s'hagi rebut una resposta des del servidor. Les respostes són enviades com arguments de la funció `jsonify()` des del codi Python. Aquesta funció converteix els objectes Python en format JSON mitjançant el mòdul `json`. Cada un d'aquests objectes és un element de la llista que conté els elements de la base de dades a partir de la consulta realitzada en el mòdul `models.py`. Quan hi ha una resposta des del servidor, el valor corresponent del gràfic s'actualitza. Finalment, s'envien els canvis al gràfic amb la funció `update()` de Chart.js.

Per actualitzar correctament els gràfics a intervals constants de temps es fa servir la funció `seInterval()`. Aquesta funció accepta dos paràmetres: la funció que s'ha de cridar i el temps que ha de passar entre crides — en mil·lèsimes de segon.

El mateix mètode es fa servir per actualitzar els valors del rellotge digital que es veu en la barra lateral. En aquest cas no obstant es crida una funció que actualitza la data, el dia de la setmana — gràcies a una llista—, i l'hora sense cridar una ruta predefinida. D'aquesta forma, tots els valors s'actualitzen en el mateix codi cada segon.

7.2. Configuració Raspberry PI i servidor

Per crear el servidor i fer accessible l'aplicació a través d'Internet s'ha aprofitat la Raspberry PI. La Raspberry Pi que s'ha fet servir per aquest projecte correspon a la versió 4 Model B i té les característiques següents:

- Processador: Broadcom BCM2711 1,50 GHz 64 bits Quad-Core Cortex-A72
- Gràfica: VideoCore VI 500,00 MHz fins a 4Kp60.

- Memòria: LPDDR4-3200 de 8 GB.
- Connectivitat: wifi a 2,40 GHz/5,00 GHz, Ethernet, Bluetooth, 2 x USB 3.0, 2 x USB 2.0, 2 x Micro-HDMI.
- Alimentació: mitjançant font externa amb connector USB-C.
- Emmagatzematge: micro-SD de 32 GB.

El sistema operatiu que s'ha instal·lat és un Debian 11 específic per a aquests tipus de plaques. S'ha triat aquest SO perquè és en el que està basat Raspbian, el sistema operatiu propi de l'empresa que fabrica aquestes plaques. En concret, s'ha triat l'última versió estable que hi ha disponible en [Gun23].

7.2.1. Configuració de l'entorn

Abans de continuar amb la configuració del servidor s'ha de preparar correctament l'entorn, ja que aquest es presenta sense interfície gràfica ni cap connexió a Internet. La configuració inicial de la placa s'ha dut a terme segons el tutorial de RaspberryTips [Ras23]. M'algrat això, per configurar tant la connexió Ethernet amb el PLC com la connexió a Internet mitjançant wifi s'han realitzat passos diferents, a causa del fet que el mètode que s'hi explica no és el més actualitzat ni el més efectiu. Per exemple, editant el fitxer `/etc/network/interfaces.d/wlan0` no és possible configurar les prioritats de xarxa, per la qual cosa la connexió a Ethernet preval sobre la connexió wifi i fa que no es pot accedir a Internet.

Per aquest motiu, les configuracions de xarxa s'han dut a terme utilitzant el programa «`nmcli`». Aquesta és una eina de terminal que permet interactuar amb NetworkManager, un servei que gestiona les configuracions de xarxa de sistemes operatius basats en Linux. Amb «`nmcli`» es pot configurar, monitorar i visualitzar l'estat de les connexions i les xarxes del sistema. El seu ús és ben senzill, i permet configurar i visualitzar els diferents paràmetres i xarxes mitjançant l'ús de comandes. Així, el problema de seguir la pàgina anterior ha estat que la prioritat de la xarxa `eth0` corresponent al cable Ethernet era més alta que la de `wlan$0$`, corresponent al wifi — com es pot observar en la imatge 7.2.

Figura 7.2.: Com es pot veure mitjançant la comanda `ipr`, la prioritat de la xarxa `eth0` és més gran que la de la xarxa `wlan0`.

La forma de configurar xarxes amb aquesta comanda és ben senzilla. Per exemple, per configurar Ethernet es pot fer:

```
sudo nmcli c add type ethernet con-name XXX ifname eth0 ip4 YYY.YYY.YY.YY/PP
    gw4 yyy.yyy.yy.yy
```

On «XXX» és el nom de la interfície, «YYY.YYY.YY.YY/PP» és la IP i el Port que se li vol assignar i «yyy.yyy.yy.yy» és la ip de la passarel·la o porta d'entrada de la connexió. En aquest cas particular no s'ha assignat cap DNS ja que mitjançant Ethernet només s'ha de realitzar la connexió amb el PLC. Per configurar una xarxa de wifi, en canvi, ho podem fer amb aquesta altra comanda:

```
sudo nmcli dev wifi con XXX password PSW
```

On «XXX» és el SSID de la wifi i PSW és la contrasenya. Per altra banda, i com s'ha explicat abans, per configurar correctament les prioritats de xarxa ho podem fer amb `ipr`.

```
sudo ip r replace default via XXX.XXX.XX.XX dev wlan0 metric YYY
```

En aquesta comanda l'opció `replace` serveix per substituir una ruta existent de la taula d'encaminament; `default` indica que la ruta que s'està modificant és la per defecte; `via192.168.0.1` indica l'adreça per on s'ha d'encaminar el tràfic de la xarxa. El paràmetre `defwlan0` fa referència al fet que el tràfic s'enviarà a través del dispositiu de xarxa `wlan0`, mentre que `metric99` estableix la seva mètrica, que es fa servir per determinar la seva prioritat — com menor sigui la mètrica, més gran serà. A la pàgina de `die.net` es pot trobar més informació sobre el funcionament intern de Linux [Die23].

Una vegada s'han configurat correctament les xarxes i determinat les seves prioritats, la connexió amb la Raspberry PI s'ha realitzat tant presencialment per teclat i ratolí com mitjançant SSH. El protocol *Secure Shell* o SSH és un protocol de xarxa que permet la connexió remota amb un altre dispositiu de forma segura perquè totes les transferències de dades estan xifrades. SSH permet l'ús de diversos protocols d'encriptació segons les necessitats dels usuaris. La forma d'accedir a la Raspberry PI mitjançant SSH és executant la següent comanda en la terminal de Debian:

```
ssh -p 22 pi@192.168.1.200
```

La IP pública de la Raspberry PI és la 192.168.1.200, l'usuari amb el qual es realitza la connexió és «`pi`» i aquesta es du a terme a través del port 22, que és el port predeterminat per a les connexions SSH.

7.2.2. Configuració del servidor web

Amb el Sistema Operatiu instal·lat, les configuracions de xarxa realitzades i establides les prioritats de les rutes és hora de triar, instal·lar i configurar un servidor que permeti accedir a l'aplicació Flask i a la pàgina web a través d'Internet. D'aquesta forma s'ha triat un servidor Apache per a tal fi, ja que s'ha fet servir en algunes assignatures i ja es coneix el seu funcionament [Fou23a].

Apache HTTP Server, també conegut com a *Apache Web Server* és, amb un 32.1% de quota de mercat — amb data d'abril de 2023 —, un dels servidors web més populars del món [Sur23].

De codi obert i disponible per diverses plataformes, suporta una gran varietat de llenguatges de programació com PHP, Python, Perl o Ruby, així com múltiples sistemes de gestió de bases de dades. Com que és un servidor software s'ha d'instal·lar en un servidor físic o virtual per poder oferir serveis web a través d'Internet [HNY99]. Això vol dir que la seva principal tasca és establir la connexió i gestionar l'enviament de dades entre el servidor físic — en aquest cas Flask — i els clients que hi accedeixen mitjançant els seus navegadors web. Per exemple, quan un usuari accedeix a un servei web, el seu programa tramet una sol·licitud al servidor mitjançant el protocol HTTP mentre Apache gestiona la consulta i la resposta des del servidor com a intermediari. Per instal·lar Apache a una màquina Debian es pot fer servir `apt`:

```
sudo apt install apache2
```

Quan s'ha instal·lat, es crea el directori `/var/www` on s'ubicaran els fitxers font de l'aplicació web. Inicialment, Apache ja conté un fitxer HTML de benvinguda per tal de testejar el seu funcionament. Per comprovar la instal·lació es pot iniciar el servidor i provar d'accedir a aquesta pàgina a través del navegador:

```
service apache2 start
```

Si s'ha engegat correctament, es pot accedir a la pàgina de benvinguda a través de la IP pública de la Raspberry PI i el port 80. Apache2 es pot configurar editant els fitxers ubicats en el directori `/etc/apache2/sites-available`. Per defecte crea dos fitxers diferents:

- `000-default.conf`: estableix el contingut web que s'ha de retornar a través del port 80, el port per defecte per a les comunicacions HTTP.
- `default-ssl.conf`: estableix la configuració per al port 443. Aquest és el port per defecte per a les sol·licituds HTTPS xifrades mitjançant SSL/TLS. Com que no és segur que aquest certificat estigui instal·lat en un inici s'hi aplica la directiva `<IfModulemod_ssl.c>`, que només executa la configuració si el mòdul `mod_ssl` està carregat i disponible. Mitjançant aquest mòdul es proporciona suport per a que les connexions estiguin xifrades d'extrem a extrem.

Per a la configuració inicial de l'aplicació s'ha editat el fitxer `000-default.conf` per a comunicacions no xifrades — més endavant, en la subsecció 7.2.2, es configura el protocol HTTPS per millorar la seguretat de l'aplicació. La configuració d'Apache2 és senzilla i permet tractar el servidor i l'aplicació web separadament. D'aquesta forma la configuració del servidor s'aplica a tot el servidor web, mentre que la configuració del lloc web només s'aplica a les aplicacions web específiques. Per configurar que l'aplicació es mostri localment a través del port 80 s'ha editat aquest fitxer de la següent forma:

```
<VirtualHost *:80>
    ServerName ingiot.local
    WSGIDaemonProcess ingiot user=<pi> group=<pi> threads=5
    WSGIScriptAlias / /var/www/ingiot/ingiot.wsgi
    <Directory /var/www/ingiot/ingiot>
        Require all granted.
    </Directory>
    ErrorLog ${APACHE_LOG_DIR}/myflaskapp-error.log
```

```
    CustomLog ${APACHE_LOG_DIR}/myflaskapp-access.log combined
</VirtualHost>
```

Es pot observar que totes les peticions que entrin mitjançant el port 80 es redirigeixen al fitxer `ingiot.wsgi`, del que se'n parla en la subsecció 7.2.2. Un cop s'ha editat la configuració, s'ha d'activar mitjançant les següents comandes.

```
A2enmod /etc/apache2/sites-available/000-default.conf
ls /etc/apache2/sites-enabled/
```

La primera comanda activa el fitxer de configuració, mentre que amb la segona es comprova que ho estigui llistant el contingut del directori corresponent. Si aquest hi és, es pot iniciar el servidor Apache2, o reiniciar mitjançant l'ordre `restart` si ja ho està.

Especificació WSGI

Web Server Gateway Interface o WSGI és una especificació que descriu la comunicació entre un servidor i una aplicació escrita en Python. Definida dins del PEP 333 [Fou22], forma part del funcionament intern d'aquest llenguatge i fa de pont entre Apache2 i Flask, convertint les sol·licituds HTTP entrants al estàndard WSGI i viceversa. Així, Flask ja conté el seu propi servidor WSGI, encara que aquest no és capaç d'entendre múltiples peticions a la vegada, té menor rendiment i el seu ús està pensat per al desenvolupament més que per a l'aplicació final. És per això que generalment es fa ús d'altres plataformes o eines, com *gunicorn*, *mod_wsgi* o *uWSGI*.

Així mateix, els servidors WSGI disposen de servidors HTTP incorporats, tot i que un servidor HTTP dedicat sol ser més segur, eficient i capaç. El fet de posar un servidor HTTP enfront d'un servidor WSGI es coneix com *proxy reverse* [Ron22]. Per l'aplicació final s'ha triat fer un de *mod_wsgi* per al servei WSGI.

El *mod_wsgi* és un servidor WSGI integrat amb Apache. Facilita la configuració i l'inici del servidor sense necessitar escriure la configuració d'Apache. En la pàgina oficial de Flask i en la llibreria *PyPI* es pot trobar més informació. Per instal·lar-lo és necessari un compilador i el servidor Apache, així com les capçaleres de desenvolupament. Per fer-ho, es pot utilitzar *pip*:

```
pip install mod_wsgi
```

La configuració d'aquesta especificació es fa mitjançant el fitxer `ingiot.wsgi` ubicat dins del directori `/var/www/`. Aquí s'ha d'indicar el directori on es troba l'aplicació web per tal de poder efectuar l'enviament de peticions correctament. D'aquesta forma, la configuració WSGI d'aquesta aplicació queda de la forma següent:

```
import sys
import logging
logging.basicConfig(stream=sys.stderr)
sys.path.insert(0, "/var/ww/ingiot/ingiot/")
from app import app as application
```

D'altra banda, també s'han d'afegir tots aquells fitxers de l'aplicació necessaris perquè es puguin executar sense problemes al mateix directori.

Seguretat de la capa de transport

La «Seguretat de la capa de transport» o TLS — per les seves sigles en anglès, *Transport Layer Security* — és el successor de *Secure Sockets Layer* o SSL. Aquest protocol criptogràfic permet assegurar les comunicacions realitzades a través d'una xarxa com Internet. Funciona intercanviant registres, els quals poden ser comprimits, xifrats i empaquetats amb un codi d'autenticació MAC. D'aquesta forma, cada registre conté un camp `content_type` que especifica el protocol de nivell superior que s'està fent servir.

D'aquesta forma, un cop iniciada la connexió el nivell de registre encapsula un altre protocol: *handshake* o «Protocol d'acord», que al seu torn conté el `content_type22`. Amb això s'aconsegueix que el tant el client com el servidor enviïn diverses estructures *handshake*:

1) Envia un missatge *ClientHello* amb una llista de tots els paràmetres de la capa, així com la versió del protocol SSL més alta disponible. També pot enviar bytes aleatoris anomenats «*Challenge* del client», així com l'identificador de la sessió.

2) El client rep un registre *ServerHello*, en el qual es trien els paràmetres de la connexió a partir de les opcions oferides anteriorment.

3) Un cop els paràmetres són coneguts client i servidor intercanvien els certificats, els quals depenen de les claus públiques de xifratge disponibles. Generalment, aquests corresponen a l'estàndard X.509, tot i que també s'especifica l'ús de certificats *OpenPGP* en un esborrany [LLC23].

4) Client i servidor negocien una clau secreta simètrica anomenada *master secret*. Això es pot fer mitjançant el protocol «Diffie-Hellman» o bé xifrant una clau secreta amb una clau pública — que serà desxifrada amb la clau privada corresponent. Totes les dades de claus restants es calculen partint d'aquest *master secret* i d'una funció pseudoaleatòria.

Això permet dotar les aplicacions web d'una robusta capa de seguretat i xifratge que no és fàcil de vèncer. En el llibre [KR21] es pot consultar les vulnerabilitats més comunes que poden patir els llocs web, mentre que *Internet Engineering Task Force* és l'organització que elabora i publica els nous estàndards TLS [LLC23].

Per encriptar l'aplicació seguint aquest estàndard s'ha fet servir l'autoritat *Let's encrypt*. Aquesta és una autoritat certificadora creada l'any 16 i que proporciona certificats X.509 de forma gratuïta a través d'un mètode automatitzat dissenyat per eliminar el complex procés de creació manual que hi havia fins aquell moment. Per crear un certificat TLS mitjançant aquesta autoritat, s'ha d'instal·lar el client *Let's Encrypt Certbot*, que s'encarregarà de renovar automàticament el certificat i indicar el nom de domini al qual se li ha d'aplicar:

```
certbot --apache -d nomdomini.com
```

S'han de seguir els passos corresponents. Per comprovar que aquest bot estigui actiu es pot fer servir `systemctl`:

```
systemctl status certbot.timer
```

Amb el certificat SSL instal·lat s'ha de configurar correctament Apache2 perquè accepti peticions HTTPS. Això es pot realitzar editant el fitxer de configuració `default-ssl.conf` que es mostra en la subsecció 7.2.2.

Amb això Apache2 ja pot escoltar peticions a través del port 443, que és el port per defecte de les peticions TLS/HTTPS. També està disponible l'opció d'afegir una redirecció amb l'objectiu que les peticions realitzades a través del port 80 hagin de passar pel 443.

Publicació de l'aplicació web

De moment, i com que l'aplicació no deixa de ser un prototip, no s'ha adquirit un nom de domini per a la seva publicació. Així, per a fer accessible l'aplicació a través de dispositius connectats a la mateixa xarxa s'ha enllaçat una IP pública amb el servidor Flask. Per fer que Flask escolti a través d'aquesta IP s'ha d'especificar dins el mètode `run`.

SystemD

Per evitar que l'aplicació falli perquè hi ha hagut un tall de corrent o s'ha reiniciat per algun motiu el dispositiu s'ha configurat *SystemD*. *SystemD* és un dimoni d'administració de sistema dissenyat pel nucli Linux. El nom ve del sufix *System Daemon*, o processos en segon pla, i va ser desenvolupat per substituir l'anterior sistema d'arrencada d'inici *init* heretat dels sistemes operatius «System V» i BSD (*Berkeley Software Distribution*).

Amb aquest sistema podem fer que les aplicacions s'iniciïn juntament amb el sistema operatiu, així com engegar, aturar, configurar o reiniciar serveis. Per configurar que una aplicació s'engegui amb el sistema s'ha d'editar el fitxer `/etc/systemd/system/ingiot.service`. *SystemD* permet múltiples formes d'engegar les aplicacions [PSa22].

8. Millores del prototip de cara a futures implementacions

En la valoració final d'aquest treball, s'ha de tindre en compte que s'ha realitzat una demostració de com podria quedar la implementació final. Amb això present, es poden observar algunes mancances en el resultat final.

En primer lloc, el més fàcil seria adaptar l'aplicació web segons els criteris únics de cada client. Com es pot veure en les captures de l'annex A, no s'ha fet un treball massa acurat ni atractiu en l'aplicació web, sinó més aviat funcional. Tampoc és clar que tots els clients necessitin una aplicació responsiva, i l'estètica que s'ha definit per a l'aplicació web és millorable si es compara amb altres sistemes creats mitjançant tecnologies modernes com Grafana. Això no obstant, no s'ha volgut aprofundir més en aquest apartat, ja que només és una demostració tècnica i cada client pot tindre les seves necessitats.

Seguidament, el càlcul de l'OEE i els seus paràmetres també podria acceptar millores. Entre els conceptes definits pel client a l'inici d'aquest projecte, cap a l'any dinou, hi havia la necessitat de poder definir fins a 5 descansos diferents. En la pràctica, en canvi, s'ha decidit prescindir d'això, ja que és una petició concreta d'un client i no tots poden tindre la mateixa necessitat. En lloc d'això, s'ha decidit definir el mínim imprescindible per a un client qualsevol, i aplicar les modificacions pertinents quan sigui necessari. Així mateix, també es podria definir una hora final per als descansos i manteniments programats. Tot i que s'ha considerat que l'aplicació no s'hauria d'aturar mai, i per aquest mateix motiu no s'ha realitzat, costaria poc d'afegir i ajudaria a considerar els temps perduts a causa dels descansos i manteniments d'una millor forma, ja que passat un determinat moment aquest càlcul hauria de ser prescindible (0 en la realitat).

Una altra de les modificacions o estudis que es podrien dur a terme a futur és el fet que tot aquest treball s'ha intentat simplificar, i seria interessant estudiar com es comporten els dos fils de comunicació, així com la classe que du a terme el càlcul de l'índex OEE en altres llenguatges. És ben sabut que Python no és molt amigable amb els recursos del sistema, i les Raspberry Pi són limitades en quant aquests, així que seria interessant traslladar aquest treball en llenguatges com C o C#, més ràpids i igual de robustos que Python.

Finalment, una de les modificacions interessants que es podria fer seria la de crear diversos bancs de treball o un de prou gran perquè simulessin una línia real de producció. Amb això es podria imitar encara més la realitat, i permetria observar millor els temps de càlcul, processat, escriptura i lectura de la Raspberry Pi. Malgrat això, i com en els casos anteriors, definir això s'ha considerat innecessari a causa del fet que Ingimec rarament dissenya dues línies similars.

9. Conclusió

El càlcul de l'OEE, i en general la creació de sistemes que puguin calcular i mostrar en temps real la producció d'una línia, és determinant per a la indústria. No solament permet estudiar i implementar millores, sinó que facilita l'estudi de cara a futur de quins són els problemes de disseny de la línia, de forma que aquests es puguin millorar. D'aquesta forma no és d'estranyar que cada vegada més empreses dedicades al disseny, programació i muntatge de línies de producció ofereixin sistemes que permetin visualitzar de formes amigables i intuïtives aquests paràmetres.

Això no obstant, i tot i que el càlcul teòric de l'OEE és senzill, la seva implementació no ho és tant. Això és degut a causa del fet que abans de realitzar el disseny i parametritzar els càlculs s'ha de tindre clar el funcionament tant de la línia com del client, de forma que es pugui predir amb suficient encert el comportament real de la producció. En concret, aquest és un problema amb el qual personalment m'he trobat, ja que quan vaig començar a fer aquest treball tenia molts dubtes sobre quins factors eren determinants per al càlcul, i quins prescindibles. Per exemple, al començament no pensava que si la disponibilitat és penalitzada degut a algun error, no es poden penalitzar la qualitat ni la productivitat. Aquest fet un cop raonat i estudiat té tot el sentit, però en un primer moment no havia estat plantejat. D'altra banda, no només és necessari saber el temps de descans que hi ha en un torn, sinó que també ho és saber com a mínim l'hora en què aquest comença. Això s'ha sabut gràcies al fet que en les proves dutes a terme es va observar que hi havia moments en què la disponibilitat tenia percentatges negatius, cosa que no hauria de passar mai. Aquest fet es produïa, ja que la forma de calcular els temps de les parades planificades era errònia i es tenia en compte durant tot el torn (fent una regla de tres). Això provocava que si un torn començava amb un error, el temps d'aquest era igual al temps total de treball fet, fet que en ajuntar-se amb el fet de restar el temps dels manteniments o parades planificades provocava resultats negatius.

Per acabar amb la valoració dels càlculs d'aquest índex i dels seus paràmetres, també s'ha de tindre en compte que no es poden comptar les peces fabricades durant períodes de manteniment o d'error, ja que no s'hauria de tindre en compte la producció ni la qualitat en aquests moments perquè ja es penalitza la disponibilitat en el càlcul. Arribats aquí, he de donar les gràcies novament al director d'aquest treball per aconsellar-me sobre aquests punts.

Pel que respecta als temes pràctics, segurament el banc de proves no es va plantejar inicialment de la millor forma possible a causa dels errors comentats en el paràgraf anterior. Per exemple ara és sabut que no era necessari definir un bit de Manteniment o d'Auditoria, ja que en aquests casos el funcionament era el mateix que si el banc hagués estat parat perquè el càlcul d'aquest índex només es té en compte en els moments en què la producció es realitza en mode automàtic.

Així mateix, s'hagués pogut aprofitar més l'HMI per jugar i provar el prototip final amb màquines que simulessin una línia de producció real. Per exemple, es podrien afegir en aquesta pantalla botons per simular Turmats, Robots o Cintes.

En conclusió, aquest treball ha servit per aprendre sobre el funcionament de la indústria i com és de necessari per a aquest el control de la producció. També m'ha ajudat a comprendre una

part important de l'enginyeria, ja que sense els enginyers de planta i els encarregats d'estudiar el seu rendiment la indústria amb tota seguretat no podria avançar a la velocitat que ho està fent.

Bibliografia

[Bri06] Thomas George Brinton. «Aplicaciones de la derivada». A: *Cálculo: una variable*. Pearson educación, 2006. Cap. 4, pàg. 292 - 297. ISBN: 9702606438.

[Cor21] IBM Corporation. *¿Qué es Ajax?* Cast. `https://www.ibm.com/docs/es/rational-soft-arch/9.6.1?topic=page-asynchronous-javascript-xml-ajax-overview`. 2021.

[dev23] The PIP developers. *Pip*. Angl. `https://pip.pypa.io/en/stable/`. 2023.

[Die23] Die.net. *ip(8) - Linux man page*. Angl. `https://linux.die.net/man/8/ip`. 2023.

[Fla20] David Flanagan. *JavaScript: The definitive guide. Master the World's MOst-Used Programming Language*. Angl. O'Reilly Media, Inc., 2020. 707 pàg. ISBN: 9781491952023.

[Fou22] The Python Software Foundation. *PEP 333 – Python Web Server Gateway Interface v1.0*. Angl. `https://peps.python.org/pep-0333`. 2022.

[Fou23a] The Apache Software Foundation. *Apache, HTTP Server Project*. Angl. `https://httpd.apache.org`. 2023.

[Fou23b] The Python Software Foundation. *Built-in Exceptions*. Angl. `https://docs.python.org/es/3/library/exceptions.html#bltin-exceptions`. 2023.

[Fou23c] The Python Software Foundation. *Built-in Functions*. Angl. `https://docs.python.org/3/library/functions.html#round`. 2023.

[Fou23d] The Python Software Foundation. *Datetime — Basic date and time types*. Angl. `https://docs.python.org/3/library/datetime.html`. 2023.

[Fou23e] The Python Software Foundation. *Encuentre, instale y publique paquetes de Python con PyPi*. Cast. `https://pypi.org`. 2023.

[Fou23f] The Python Software Foundation. *Errors and Exceptions*. Angl. `https://docs.python.org/es/3/tutorial/errors.html`. 2023.

[Ger16] Mark Gerwitz. *GNU kWindows*. Angl. `https://mikegerwitz.com/2016/04/gnu-kwindows`. 2016.

[GS13] Molenaar Gijs i Preeker Stephan. *Welcome to Python-Snap7's documentation*. Angl. `https://python-snap7.readthedocs.io/en/latest/index.html`. 2013.

[Gun23] a Debian Developer Gunnar Wolf. *Tested images*. Angl. `https://raspi.debian.net/tested-images/`. 2023.

[HNY99] Yiming Hu, A. Nanda i Qing Yang. «Measurement, analysis and performance improvement of the Apache Web server». A: *1999 IEEE International Performance, Computing and Communications Conference (Cat. No.99CH36305)*. Institute of Electrical i Electronics Engineers, 1999, pàg. 261 - 267. DOI: 10.1109/PCCC.1999.749447.

[Ing] Ingimec. *Millores Proto (document intern)*. https://github.com/6q4598/TFG/tree/main/PreviousWork/Documents/Office.

[KR21] James F. Kurose i Keith W. Ross. *Computer Networking: A Top-Down Approach*. Angl. Pearson Education Limited, 2021. 800 pàg. ISBN: 9781292405469.

[LLC23] IETF Administration LLC. *Internet Engineering Task Force*. Angl. https://www.ietf.org. 2023.

[Mic22a] Microsoft. *Preventing Cross-Site Request Forgery (CSRF) Attacks in ASP.NET MVC Application*. Angl. https://learn.microsoft.com/en-us/aspnet/web-api/overview/security/preventing-cross-site-request-forgery-csrf-attacks. 2022.

[Mic22b] Microsoft. *Windows Subsystem for Linux documentation*. Angl. https://learn.microsoft.com/en-us/windows/wsl. 2022.

[Mic23] Microsoft. *Bandwidth pricing*. Angl. https://azure.microsoft.com/en-us/pricing/details/bandwidth/. 2023.

[MM13] Juan A Martin-García i Rafael Mateo Martínez. *Barreras y facilitadores de la implantación del TPM*. Cast. Ed. d'Intangible Capital. https://www.redalyc.org/pdf/549/54928893011.pdf. 2013.

[Nak89] Seiichi Nakajima. *TPM Development Program. Implementing total productive maintenance (English and Japanese Edition)*. english, japan. Productivity Press, 1989. 403 pàg. ISBN: 9780915299379.

[Nar] David Nardella. *Step7 Open source Ethernet Communication Suite*. Angl. https://snap7.sourceforge.net.

[PSa22] Lennart Poettering, Kay Sievers i altres. *SystemD System and Service Manager*. Angl. https://www.freedesktop.org/wiki/Software/systemd/. 2022.

[Ras23] RaspberryTips. *How to Install Debian on Raspberry Pi (Illustrated guide)*. Angl. https://raspberrytips.com/install-debian-on-raspberry-pi/. 2023.

[Ron22] Armin Ronacher. *Flask, web development, one drop at a time*. https://flask.palletsprojects.com/en/2.2.x/. 2022.

[Sur23] W3Techs - Web Technology Surveys. *Usage statistics of web servers*. Angl. https://w3techs.com/technologies/overview/web_server. 2023.

[Tea] The Bootstrap Team. *Examples sidebars*. Angl. https://getbootstrap.com/docs/5.0/examples/sidebars/.

[Tea23] The Bootstrap Team. *Build fast, responsive sites with Bootstrap*. Angl. https://getbootstrap.com/docs/5.3/getting-started/introduction/. 2023.

[W3S23] W3Schools. *Ajax introduction*. Angl. https://www.w3schools.com/js/js_ajax_intro.asp. 2023.

Part II.

Apèndixs

A. Dipòsit del projecte

Durant aquest projecte s'ha fet servir un sistema de control de versions *git*. En concret, s'ha fet servir GitHub per a tal fi. Git és un sistema de control de versions distribuït que permet el seguiment dels canvis realitzats, així com la col·laboració entre projectes. Per altra banda, GitHub és una plataforma web d'allotjament i gestió de projectes basada en Git. Entre altres coses, GitHub permet als usuaris crear, allotjar i col · laborar en projectes de programari lliure. D'aquesta forma, tots els codis i la documentació d'aquest treball s'han inclòs dins del dipòsit de GitHub `6q4598/TFG`.

Aquest repositori s'ha estructurat de la següent forma.

A.1. `Flask`

Conté el servidor Flask, així com la pàgina web i tots els seus fitxers. Està estructurat mitjançant els següents directoris:

- `Static`: inclou tots els fitxers, imatges i recursos que es necessiten per a que la pàgina web funcioni correctament. Conté directoris on ubicar els codis CSS, les imatges, els codis JavaScript i les bases de dades de test.
- `Templates`: inclou els codis HTML de la web.

El codi del servidor està directament ubicat a dins d'aquest directori i no en una subcarpeta.

A.2. `PreviousWork`

Guarda tota la investigació prèvia sobre .NET, MVC, Dockers, Cosmos DB etc. Al final es va decidir fer-ho tot amb Flask i Python, amb una Raspberry PI, pel que això es va rebutjar.

Aquest directori està estructurat de la següent manera:

- `ApiNet`: conté tots els projectes que es van crear de nou quan encara s'intentava fer tot mitjançant .NET i MVC.
  - `IOT4246`
  - `IOT4246_mvc`
  - `Screenshots`
- `ArduinoCode`: amb els codis de la ESP32 que es van fer en el seu moment. Aquests es van programar mitjançant Arduino IDE.
  - libraries
  - sketchEsp32Linux

- `BackupsDatabase`: on s'ubiquen les còpies de seguretat de les bases de dades.
- `Documents`: Documents i treballs obert que m'han servit de referència.
- `Esp32LinuxSockets`: En cert moment, es va voler fer un programa que des de la Raspberry PI rebés les dades enviades des de l'ESP32 i ho guardés en una base de dades relacional.

Dins aquest directori també s'inclouen alguns apunts interessants sobre Dockers i altres tecnologies, així com esborranys d'aquesta memòria.

A.3. `RaspyCom`

Amb els fils de comunicació amb el PLC, i d'escriptura a la base de dades. També conté la classe que fa el càlcul de l'OEE i tots els seus paràmetres.

- `util`: és l'única carpeta que conté aquest directori, i és el calaix de sastre on s'han guardat totes les proves que s'han realitzat.
- `main.py`
- `oee.py`
- `db.py`

B. Mapejat memòria del DB 100

En el DB 100 s'han definit els següents bits — taula B — segons el que es va decidir en l'origen del projecte:

Descripció	Tipus	Byte	Bit
Error fusibles 24v exterior	Bool	0	0
Error fusibles 24v seguretat	Bool	0	1
Fallo Modulo emergencias	Bool	0	2
Fallo Modulo Barreras	Bool	0	3
Fallo Puertas	Bool	0	4
Fallo Termico Bomba Encebado	Bool	0	5
Fallo Presostato	Bool	0	6
Dispositivo profinet IFM en error 204KF1	Bool	0	7
Dispositivo profinet SMC en error 206K1	Bool	1	0
Conexión PC perdida	Bool	1	1
Test anulado por el usuario	bool	6	2
Modelo no corresponde parametros introducidos	bool	6	3
Medidas bomba incorrecta	bool	6	4
Error lectura lector intensidad 230	Bool	9	7
Error lectura lector intensidad 400	Bool	10	0
Error lectura variador 230v	Bool	10	1
Error lectura variador 400v	Bool	10	2
Error escritura lector intensidad 230	Bool	10	3
Error escritura lector intensidad 400	Bool	10	4
Error escritura variador 230v	Bool	10	5
Error escritura variador 400v	Bool	10	6
Variador en error 230	Bool	10	7
Variador en error 400	Bool	11	0
Cilindro Cierre Vertical A1.1 - Tiempo máximo movimiento a origen	Bool	12	3
Cilindro Cierre Vertical A1.1 - Tiempo máximo movimiento a trabajo	Bool	12	4
Cilindro CierreVertical A1.1 - Sensores detectan a la vez	Bool	12	5
Cilindro CierreVertical A1.1 - Posición perdida	Bool	12	6
Cilindro CierreHorizontal A1.2 - Tiempo máximo movimiento a origen	Bool	12	7
Cilindro CierreHorizontal A1.2 - Tiempo máximo movimiento a trabajo	Bool	13	0

Descripció	Tipus	Byte	Bit
Cilindro CierreHorizontal A1.2 - Sensores detectan a la vez	Bool	19	7
Cilindro CierreHorizontal A1.2 - Posición perdida	Bool	20	0
Cilindro CierreHorizontalPurgador A1.3 - Tiempo máximo movimiento a origen	Bool	27	0
Cilindro CierreHorizontalPurgador A1.3 - Tiempo máximo movimiento a trabajo	Bool	27	1
Cilindro CierreHorizontalPurgador A1.3 - Sensores detectan a la vez	Bool	27	2
Cilindro CierreHorizontalPurgador A1.3 - Posición perdida	Bool	27	3
Cilindro DesplazamientoCarro A1.4 - Tiempo máximo movimiento a origen	Bool	27	4
Cilindro DesplazamientoCarro A1.4 - Tiempo máximo movimiento a trabajo	Bool	27	5
Cilindro DesplazamientoCarro A1.4 - Sensores detectan a la vez	Bool	27	6
Cilindro DesplazamientoCarro A1.4 - Posición perdida	Bool	27	7
Cilindro BloqueoCarro A1.5 - Tiempo máximo movimiento a origen	Bool	28	0
Cilindro BloqueoCarro A1.5 - Tiempo máximo movimiento a trabajo	Bool	28	1
Cilindro BloqueoCarro A1.5 - Sensores detectan a la vez	Bool	28	2
Cilindro BloqueoCarro A1.5 - Posición perdida	Bool	28	3
Cilindro Puerta A1.6 - Tiempo máximo movimiento a origen	Bool	28	4
Cilindro Puerta A1.6 - Tiempo máximo movimiento a trabajo	Bool	28	5
Cilindro Puerta A1.6 - Sensores detectan a la vez	Bool	28	6
Cilindro Puerta A1.6 - Posición perdida	Bool	28	7
Valvula Entrada AH01 A2.1 - Tiempo máximo movimiento a origen	Bool	29	0
Valvula Entrada AH01 A2.1 - Tiempo máximo movimiento a trabajo	Bool	29	1
Valvula Entrada AH01 A2.1 - Sensores detectan a la vez	Bool	29	2
Valvula Entrada AH01 A2.1 - Posición perdida	Bool	29	3
Valvula Entrada AH02 A2.2 - Tiempo máximo movimiento a origen	Bool	29	4

Descripció	Tipus	Byte	Bit
Valvula Entrada AH02 A2.2 - Tiempo máximo movimiento a trabajo	Bool	29	5
Valvula Entrada AH02 A2.2 - Sensores detectan a la vez	Bool	29	6
Valvula Entrada AH02 A2.2 - Posición perdida	Bool	29	7
Ejecución alcanzado	Bool	30	0
Falla definida por el usuario 1	Bool	30	1
Fallo definido por el usuario 2	Bool	30	2
Tiempo de encendido alcanzado	Bool	30	3
Pérdida de carga	Bool	30	4
Feedback PID perdido durante la carrera	Bool	30	5
Tiempo de espera límite de corriente rápida	Bool	30	6
Fallo esclavo en la sincronización de velocidad	Bool	30	7

Taula B.1.: Taula que mostra els bits definits en el DB 100.

C. Captures de pantalla de l'aplicació web

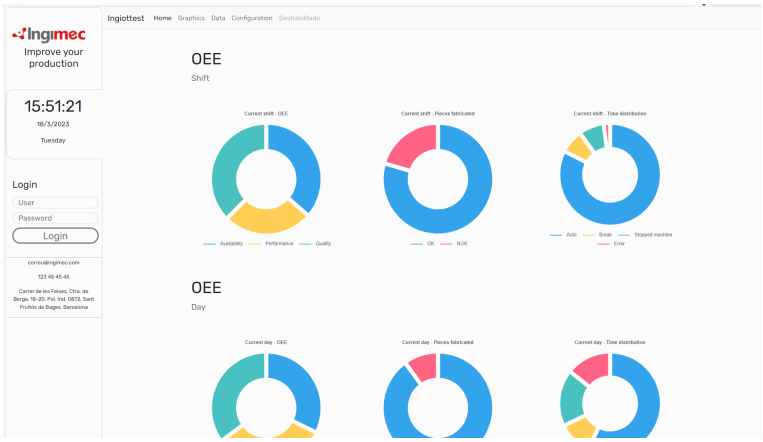

Figura C.1.: Captura 1. Es mostren els valors calculats tant per al torn com per al dia actuals.

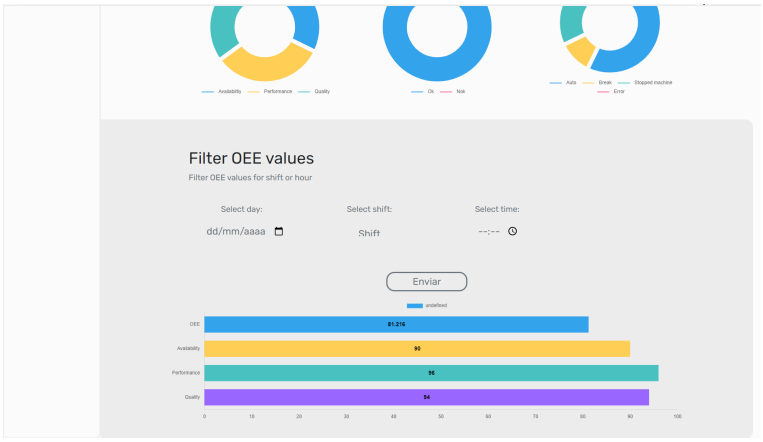

Figura C.2.: Captura 2. Es permet consultar els valors de l'OEE per a un torn qualsevol de la base de dades.