Universitat Politècnica de Catalunya (UPC)

Treball fi de Màster

Sistema per a la gestió dels tiquets de compra virtuals

Autor David Aleu Moseguí

Director Marc Alier Forment

Codirectora Maria José Casany Guerrero

Barcelona, 21 de juny de 2021

David Aleu Moseguí
Sistema per a la gestió dels tiquets de compra virtuals
21 de juny de 2021
Director: Marc Alier Forment
Codirectora: Maria José Casany Guerrero
Universitat Politècnica de Catalunya

Màster en Enginyeria Informàtica
Facultat d'informàtica de Barcelona (FIB)
C/Jordi Girona, 1
08034 and Barcelona

Resum

En el nostre dia a dia, en el moment que efectuem una compra en un comerç físic, rebem un tiquet com a comprovant d'aquesta. Una gran part d'aquests tiquets estan fabricats d'un material no reciclable, el qual comporta que es generi una gran quantitat de residus. Aquest projecte consisteix en crear un sistema, on els comerços puguin efectuar totes les seves vendes sense la necessitat d'imprimir el tiquet, i que els clients els puguin rebre fàcilment en un dispositiu mòbil.

Resumen

En nuestro día a día, en el momento que efectuamos una compra en un comercio físico, recibimos un ticket como comprobante de esta. Un gran parte de estos tickets están fabricados por un material no reciclable, el cual conlleva que se genere un gran cantidad de residuos. Este proyecto consiste en crear un sistema, donde los comercios puedan efectuar todas sus ventas sin la necesidad de imprimir el ticket, i que los clientes los puedan recibir fácilmente en un dispositivo móvil.

Abstract

Every day, whenever we make a purchase in a physical store, we receive a ticket as a proof of this action. A big amount of these tickets is made of a non-recyclable material, which leads to generate a large amount of waste. This project consists of creating a system, where the businesses can make all their sales without the need to print the ticket, and where the customers can easily receive the on a mobile device.

Agraïments

Agraeixo al Marc Alier Forment i a la Maria José Casany Guerrero haver acceptat ser els directors del meu Treball de Final de Màster, els seus consells, ajuda i supervisió durant el projecte.

Agraeixo als meus companys de la universitat, per fer que el transcurs del desenvolupament d'aquest projecte hagi estat més fàcil i agradable.

Finalment, agraeixo a tota la meva família el seu suport i confiança que m'han donat durant tot el període acadèmic.

Índex

Índex de figures

Índex de taules

1 Introducció i Contextualització

1.1 Contextualització

1.1.1 La utilització dels tiquets

El tiquet (o factura simplificada) es genera just després de fer una compra en un establiment, i serveix com a comprovant d'aquesta. Gràcies a ell, podem canviar un article que ja s'hagi comprat, presentar reclamacions a l'establiment, o veure l'IVA que s'ha pagat en aquella compra. També dir, que és la única garantia que el client té en una compra física.[1][2]

Els tiquets estan formats pels següents elements:

1. Número i serie correlativa
2. Data d'expedició
3. Nom de l'establiment i la direcció
4. NIF de l'expedidor
5. Descripció del servei / desglossament de cada article o servei prestats
6. Tipus d'imposicions aplicades
7. Descomptes que puguin existir
8. Import de la compra (amb IVA)

Tot i la utilitat dels tiquets, moltes vegades no els agafem o els tirem a la primera paperera que trobem en sortir de l'establiment, ja que la seva utilitat dura poc temps o no la trobem necessària. Per exemple, quan compres menjar o vas al cinema, la quantitat que s'ha pagat normalment és baixa, i la garantia dels productes o serveis que has comprat s'acaba molt de pressa, el que fa que la utilitat del tiquet duri poc temps, i acaba essent un residu innecessari.

Per altra banda, hi ha moltes vegades que sí que ens interessa conservar-lo, com quan comprem un ordinador, ja que normalment ens serveix com a garantia d'aquest.

Tant si fem ús o no del tiquet, aquest sempre s'imprimeix, i acaba essent un residu que causa un impacte mediambiental. Per altra banda, també hi han estudis que demostren que un dels components del paper térmic (el material que composa els tiquets) són perjudicals per la salut.

Per explicar una mica quin és l'impàcte mediambiental i l'impacte en la salut de l'ús dels tiquets de compra, m'he basat en l'informe **Skip The Slip**[3], el qual ens explica el consum del paper tèrmic als Estats Units.

1.1.2 Impacte mediambiental

Només el 2019, s'ha consumit 280000 tones de paper tèrmic utilitzat als Estats Units. L'impacte mediambiental que s'estima per crear tot aquest paper, és el següent:

Boscos

La tala de 3630000 arbres anuals. Els boscos que hi ha arreu del món, col·laboren en l'eliminació d'una gran part del CO^2 causat per les activitats humanes. La desforestació, causa que cada vegada s'eliminin menys CO^2.

Aigua

El consum de $3.76 * 10^{10}$ litres d'aigua neta anuals, equivalent al consum d'aigua de 7 milions de rentadores a l'any. Avui en dia, prop de 783 milions de persones no tenen accés a aigua potable, i va en augment. L'ús d'aquesta aigua per la producció del paper tèrmic, contribueix al fet que cada vegada més gent no tingui accés a fonts potables.

Emissions de CO^2

L'emissió de $2.32 * 10^9$ kilograms de gasos hivernacle anuals, que bé a ser el mateix que generen 464000 cotxes a l'any. Totes aquestes emissions són causades per l'extracció de materials, la producció del paper tèrmic i les emissions del transport i la distribució d'aquest.

Recordar, que a causa de les emissions de CO^2, ens trobem amb el problema del canvi climàtic, responsable de l'augment de la temperatura en la Terra, l'augment del nivell del mar, recurrència de fenòmens meteorològics extrems, etc.

Ús d'energia

El consum de $9.09 * 10^{15}$ joules d'energia anuals, equivalent al consum de 10 milions de neveres a l'any. Gran part d'aquesta energia, prové de fonts no renovables, tal com el carbó o el gas natural. Al ser fonts d'energia que s'extreuen a partir de la combustió, implica un gran alliberament de CO^2 a l'atmosfera.

Residus sòlids

Es genera 149685 tones de residus provinents de la producció, el que és equivalent als residus que generen 75 milions de persones per dia. Aquests residus afecten directament al medi ambient, causant una contaminació dels ecosistemes.

1.1.3 Impacte en la salut

El paper tèrmic, a diferència del paper normal, s'imprimeix mitjançant la calor, ho sigui, no necessita tinta. Per aconseguir això, el paper porta un component químic anomenat Bisfenol-A (BPA), el qual és tòxic, i es transfereix només amb el tacte. Això vols dir, que cada vegada que agafem un tiquet de compra d'un establiment, ni que sigui amb poca quantitat, absorbim una petita part d'aquest component.

El Bisfenol-A, afecta el desenvolupament fetal i està relacionat amb el deteriorament reproductiu, la diabetis de tipus 2 i afectacions a les tiroides. A més, les persones que treballen amb tiquets de compra en el seu dia a dia, solen tenir concentracions més altes en el seu cos.

A causa d'aquest problema alguns països han pres mesures. Per exemple, des de gener de 2020, la Unió Europea va prohibir la comercialització del paper tèrmic amb més d'un 0.02% de Bisfenol-A dins de l'espai Europeu.[4][5]

1.1.4 Alternatives al tiquet físic

Algunes de les solucions que hi ha ara mateix, són les següents:

1. Només imprimir el tiquet en cas que el client ho demani a l'establiment. En bars i restaurants, normalment no necessites el comprovant.

2. Imprimir el tiquet en paper que no sigui tòxic, tot i que això continua causant un gran impacte en el medi ambient.

3. Digitalitzar el tiquet, què és l'opció en la qual em centro en aquest treball.

1.2 Objectius

1.2.1 Objectiu Principal

L'objectiu principal del projecte, consisteix en proporcionar una solució alternativa als tiquets de caixa impresos tèrmicament, per una solució paperless, ho sigui, una solució on no s'utilitzi paper.

1.2.2 Objectius Específics

- Crear un sistema, que es pugui instal·lar fàcilment, i que s'adapti als diferents sistemes que utilitzen els comerços actuals.
- Els clients dels comerços han de poder rebre el tiquet als seus dispositius movil.
- El tiquet ha d'estar certificat digitalment, per tal de poder validar la seva procedència.

1.3 Motivació

En els anys que porto en el món de la informàtica sempre he anat adquirint nous coneixements, des de les línies de codi més bàsiques fins a crear un software complet. Tot i això, en aquest sector, les tecnologies avancen contínuament, el qual comporta que hagis d'estar sempre al dia.

Donat que aquest és el treball de final de màster, considero que és un bon moment per posar en pràctica algunes de les tecnologies apreses durant aquests dos anys i fer un projecte que les combini una mica entre si.

Per altra banda, ens trobem en una crisi climàtica, la qual, en gran part, és provocada per la societat de consum. Aquest consum insostenible, comporta un malbaratament dels recursos naturals i una gran quantitat de generació de residus.

Com que aquest tema és molt gran, vaig decidir centrar-me en una petita part, els tiquets. En qualsevol comerç, sempre que comprem algun producte, s'imprimeix un tiquet (o factura simplificada) com a comprovant de la gestió que s'acaba de dur a terme. Si més no, la majoria d'aquests tiquets no són reciclables, i això comporta que generin una gran quantitat de residus.

Donat el problema que comporten els tiquets, i els coneixements que he adquirit durant el màster, va sorgir la idea de crear una impressora virtual. Per fer això, s'utilitzaria una Raspberry Pi 3 configurada com a servidor d'impressora, però en el moment que es rep el tiquet per imprimir, la Raspberry el pujaria al núvol, i l'usuari el podria descarregar mitjançant un codi QR, evitant així l'ús innecessari del paper.

1.4 Estat de l'Art

Per veure la necessitat de la implementació d'aquest projecte, s'ha dut a terme un petit estudi de mercat on es reflecteixen les principals formes de generar tiquets digitals actualment. Aquests són alguns dels negocis dels negocis que han començat a digitalitzar els tiquets.

Lidl

Lidl sembla que ha començat a fer la transició. Quan fas qualsevol compra, el tiquet sempre s'imprimeix a caixa. Tot i això, si tens la seva aplicació mòbil sempre el reps allí directament després de la compra.[6]

Decathlon

Decathlon, ha afegit l'opció del tiquet digital en els seus establiments. En el moment que compres alguna cosa, Decathlon et dóna l'opció d'imprimir el tiquet o rebre'l per correu.[7]

Mango

De la mateixa forma que Decathlon, Mango també dóna l'opció de tiquet digital en el cas que ho prefereixis.[8]

No Mas Tickets

No Mas Tickets, és una plataforma que s'utilitza per centralitzar tots els tiquets en una sola aplicació. Quan compres en un establiment que està adherit a la plataforma, reps el tiquet directament al teu perfil de l'aplicació.[9]

Tal com es pot veure en les empreses que s'han esmentat anteriorment, comença a haver-hi una transició del tiquet físic al tiquet digital. Tot i això, totes les empreses que he trobat, opten per enviar un correu electrònic o enviar els tiquets a través de l'aplicació mòbil.

Tot i que ens trobem en un món on la tecnologia està per tot arreu, no tothom en sap fer ús, per tant, el tema d'enviar els tiquets per correu pot ser un problema per aquestes persones.

Amb el projecte que es proposa, només cal disposar d'un telèfon intel·ligent per tal de poder escanejar codis QR. En el moment que el client tingui un problema per rebre el tiquet, un dependent el pot ajudar de forma quasi immediata. Si el client no té ni correu electrònic, el dependent no té temps en ensenyar-li com rebre el tiquet.

2 Metodologia

Per dur a terme el projecte, he aplicat una aproximació de la metodologia *Agile*. Aquesta, consisteix en desenvolupar el software d'una manera iterativa i incremental, ho sigui, que el projecte es divideix en varis *Sprints*, on cada un d'ells consistirà en desenvolupar una o més funcionalitats noves.

El projecte es divideix en vàris *Sprints*, on cada una d'elles aporta una part funcional nova. Això implica que al final de cada iteració, es té un software usable, i (dins del possible) que funcioni de forma completament individual a les altres parts desenvolupades.

A més, per definir cada *Sprint*, s'utilitzarà el que s'anomena història. Una història representa una funcionalitat del sistema o una necessitat del desenvolupador per tal de tirar el projecte endavant. Si una història és molt gran (história épica), aquesta es pot dividir en històries més petites. A més, cada una de les histórias pot tenir una prioritat assignada, la qual ens serveix per ordenar quines són les taques que s'han de fer abans.

Aquest projecte, inicialment s'ha dividit en 5 *Sprits* de dues setmanes cada un, els quals consistien en implementar una TPV (Terminal Punt de Venda), però donat que no es tenia el coneixement necessari per implementar el projecte, es van cancel·lar els dos últims *Sprints* per donar lloc a 4 *Sprints* nous, on es va implementar el projecte actual, la implementació d'una *RaspberryPi* com a impressora.

Normalement, en els projectes basats en metodologies agils, s'utilitzen taulells on es marca l'estat de les histórias en cada moment. Aquest taulell consta de 5 columnes, les quals indiquen l'estat de les histórias:

- Backlog: Conté totes les històries que s'han de dur a terme per realitzar el projecte.
- To Do: Històries que s'han de dur a terme en l'*Sprint* actual.
- In Progress: Històries que s'estan implementant en l'*Sprint* actual.
- Pending: Històries que estan a l'espera de ser validades en l'*Sprint* actual.
- Done: Històries que ja implementades i validades.

Inicialment, quan es crea una nova història, aquesta es posa a la primera columna del taulell (*Backlog*). En el moment que s'ha de començar un nou *Sprint*, es planifiquen totes les històries que es volen dur a terme, i es mouen a la columna de *To Do*. En el moment que es comença una història, aquesta passa a la columna de *In Progress*. Quan aquesta s'acaba, es passa a la columna de *Pending*, i seguidament, es testeja. Si la història té errors, es torna a passar a la columna de *In Progress*. Altrament, si l'història compleix amb els seus requisits i funciona correctament, es mou a la columna de *Done*.

Per tal de portar el control de totes les tasques del projecte, hi ha diversos softwares que ens poden ajudar, com ara Asana o Trello. En el cas d'aquest projecte s'ha utilitzat Trello, ja que és gratuït i és molt fàcil d'utilitzar.

3 Aspectes Tècnics

3.1 Funcionament actual

Actualment, per imprimir els tiquets en els diferents comerços, es disposa de dos components claus:

1. **TPV (Termina Punt de Venda)**: El TPV és un dispositiu que ajuda a gestionar les vendes dels diferents locals. En el moment que un client vol comprar o llogar alguna cosa en un establiment comercial, s'ha de passar per caixa. A la caixa, el venedor comença a comptabilitzar tots els elements que el client vol comprar en el TPV. Normalment, els TPV, mantenen una gestió de l'estoc de l'establiment, i els valors de caixa que hi ha en tot moment.

2. **Impressora**: La impressora és l'encarregada d'imprimir el tiquet una vegada s'han registrat tots els elements al TPV. Quan el venedor ha acabat de registrar tots els elements i el client ha pagat, des del TPV s'ordena imprimir el tiquet (comprovant de la compra). Finalment, aquest s'imprimeix i es lliura al client.

En el següent esquema es mostra el flux per aconseguir el tiquet en un establiment comercial:

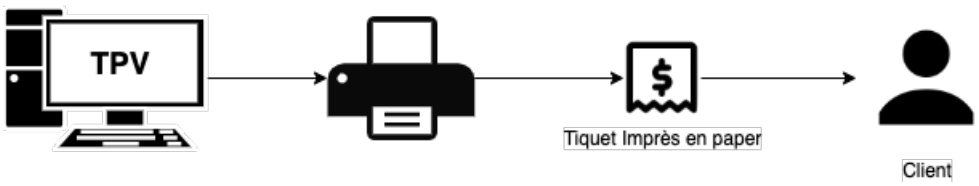


Fig. 3.1: Sistema d'impressió de tiquets actualment

3.2 Propòsit

Per tal de poder eliminar els tiquets impressos en paper del flux actual, s'implementarà un sistema el qual s'anomenarà *Wetick*.

Si ens fixem en l'esquema de l'apartat anterior, hi ha dos elements els quals són immutables: el TPV i el client. El TPV gestiona tota la informació del comerç, i això no ens afecta a l'hora

d'imprimir el tiquet. Per altra banda, si no hi ha client, no hi ha tiquet que s'hagi d'imprimir. Per tant, el sistema consistirà en substituir la impressora i el tiquet imprès en paper per altres components que ens duguin al mateix resultat: que el client rebi el tiquet de la compra que ha efectuat.

Per tal que el sistema cumpleixi amb aquesta funcionalitat, s'han dissenyat tres passos principals.

1. En primer lloc, s'haurà de configurar un dispositiu per tal que es pugui detectar com a impressora. D'aquesta manera, el sistema *Wetick* passa a ser transparent per al TPV.

2. Aquest dispositiu, haurà de ser capaç de generar el tiquet en un lloc que pugui ser accessible per als usuaris (en aquest cas, a priori s'ha pensat en el *cloud*), en el moment que rebi una ordre d'impressió.

3. Una vegada el tiquet s'ha generat, el client ha de ser capaç rebre'l d'alguna forma.

En els següents esquemes, es plasma el disseny de *Wetick*:

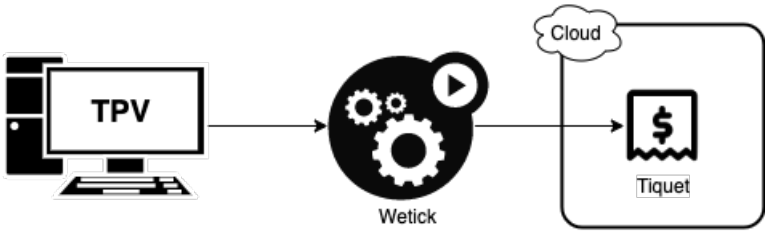


Fig. 3.2: Disseny Wetick 1

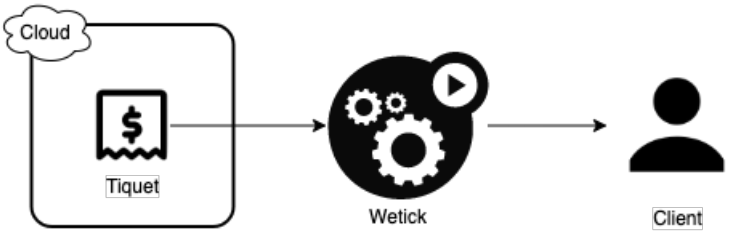


Fig. 3.3: Disseny Wetick 2

Tal com es pot veure en aquest esquema, *Wetick* tindrà dues funcions principals:

1. Transportar el tiquet generat pel TPV al *cloud*.

2. Transportar el tiquet del *cloud* al client.

3.3 Requisits Funcionals

Per tal de satisfer el disseny anterior, s'han especificat els següents requisits funcionals:

- El sistema ha de permetre detectar el dispositiu com una impressora.
- El sistema ha de permetre que els tiquets es firmin amb un certificat digital (d'aquesta manera, es pot comprovar l'autenticitat del comerç que ha expedit el tiquet).
- El sistema ha de permetre que el tiquet es guardi al *cloud*.
- El sistema ha de permetre que el tiquet es pugui descarregar des de qualsevol telèfon intel·ligent.

3.4 Requisits No Funcionals

De la mateixa manera, els requisits no funcionals del projecte són els següents:

- Privacitat: Les dades del tiquet, només es podran accedir pel client i pel comerç.
- Concurrència: El sistema ha de poder funcionar des de vàris dispositius a la vegada.
- Portabilitat: Per poder descarregar els tiquets, es necessitarà connexió a internet, i s'haurà de disposar d'un dispositiu mòbil.
- Seguretat: Els tiquets només es podran pujar des del dispositiu que estigui autenticada al servei de *cloud*.
- Usabilitat: El sistema s'ha de poder utilitzar intuïtivament.
- Eficiència: El sistema ha de gestionar el tiquet el més ràpid possible, per tal que el client el pugui descarregar.

3.5 Històries

Per tal d'organitzar totes les tasques que s'han de dur a terme per tal de complir amb els requisits, el projecte s'ha dividit en les següents set històries èpiques:

- Configurar el dispositiu com servidor d'impressions.
- Capturar els tiquets que s'imprimeixen i convertir-los a PDF.
- Firmar els tiquets amb un certificat digital.
- Pujar els tiquets a un servei de *cloud* amb un URL que no es pugui desxifrar.
- Crear una aplicació web la qual s'utilitzarà perquè el client pugui rebre el tiquet.
- Configurar un canal de comunicació entre el *backend* (lloc on es gestiona el tiquet) i el *frontend* (canal que s'utilitzarà per que l'usuari pugui rebre el tiquet).
- Muntar l'aplicació damunt d'un *Docker*.

4 Arquitectura del Sistema

A partir de tota la informació que s'ha generat en l'apartat anterior, s'ha dissenyat la següent arquitectura. Aquesta arquitectura, consisteix en una part de *hardware* i una part de *software*.

4.1 Arquitectura del Hardware

L'arquitectura del *hardware* està creada per 4 components:

1. **TPV**: El qual ve donat per pels requeriments del comerç.
2. **Dispositiu**: El dispositiu serà l'encarregat de gestionar tots els tiquets provinents del TPV.
3. **Pantalla**: Encarregada de lliurar el tiquet al client.
4. **Servei *cloud***: Lloc on es guardarà el tiquet per tal de no imprimir-lo físicament.

El *TPV* i la pantalla es connectaran directament al dispositiu encarregat de gestionar els tiquets. Per altra banda, el dispositiu es connecta al servei de *cloud* a través d'internet. En el següent diagrama es pot veure les connexions:

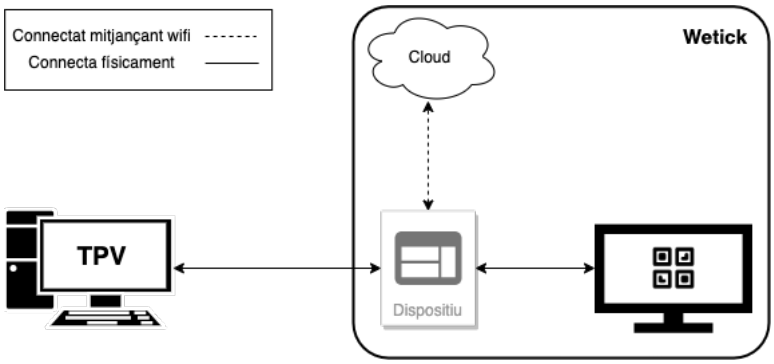


Fig. 4.1: Diagrama del hardware

A partir d'aquí, tot el software es troba dins del component **dispositiu**.

4.2 Arquitectura del Software

L'arquitectura del software està creada pels següents elements:

1. **Ordre d'impressió**: Ordre d'impressió que conté el tiquet. Aquesta s'ha generat en el TPV.

2. **Servidor**: Encarregat de gestionar el tiquet.

3. **Aplicació web**: Encarregat d'informar el tiquet al client.

El servidor sempre està atent a les possibles ordre d'impressió que puguin arribar. Quan detecta una ordre, notifica a l'aplicació web que el document s'ha començat a gestionar. El gestiona, el puja al servei *cloud* i finalment torna a notificar l'aplicació web amb l'URL. L'esquema corresponent seria el següent:

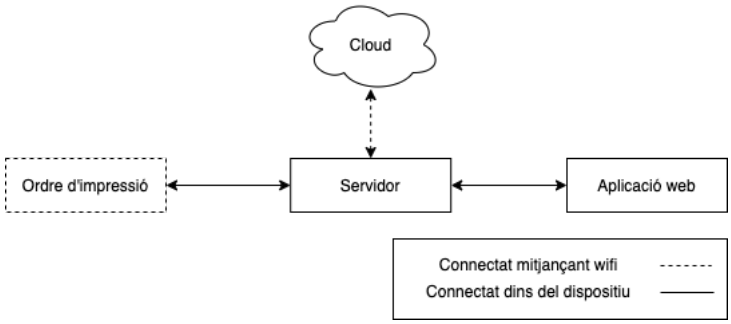


Fig. 4.2: Diagrama del software

4.2.1 Servidor

El servidor és la base del projecte, i és el que conté les connexions amb totes les altres parts del software.

Aquest, està compost pels següents mòduls, que també estan connectats entre ells:

1. **file_listener**: Encarregat de capturar totes les ordres d'impressió.

2. **file_sign**: Encarregat de certificar el tiquet.

3. **file_uploader**: Encarregat de pujar el tiquet al servei *cloud*.

4. **socket**: Encarregat de notificar l'estat dels tiquets a l'aplicació web.

El servidor està comunicat seguint el següent diagrama:

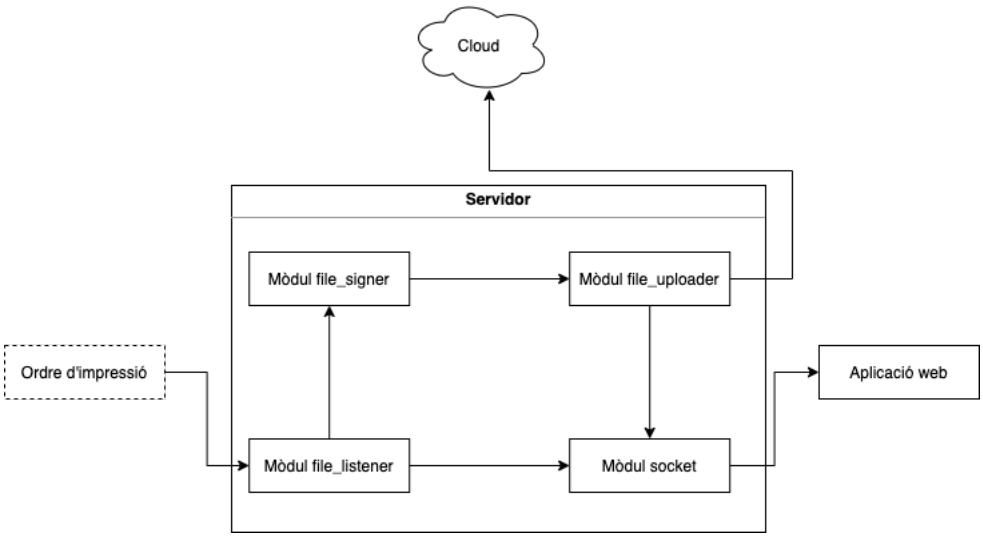


Fig. 4.3: Diagrama del servidor

Quan el mòdul de *file_listener* detecta una nova d'ordre d'impressió, aquest envia el tiquet al módul *file_signer*. Al mateix temps, també notifica al mòdul *socket* informant que s'està gestionant un nou tiquet, i aquest notifica l'aplicació web.

Seguidament, el mòdul *file_signer*, certifica el document amb un certificat digital, i l'envia al mòdul *file_uploader*.

A continuació, el mòdul *file_uploader*, puja el document al servei *cloud*, li assigna permisos de lectura, i agafa l'URL. Seguidament, notifica el mòdul *socket* amb l'URL del document que s'ha gestionat.

Finalment, el mòdul *socket*, envia l'URL a l'aplicació web, s'utilitzarà per informar el tiquet al client del comerç.

4.2.2 Aplicació Web

Finalment, l'arquitectura de l'aplicació web, només consta de dos elements:

1. **app.service**: Encarregat de capturar tots els missatges provinents del mòdul *socket* del servidor.

2. **app.component**: Encarregat de mostrar els missatges que rep *app.service* per pantalla.

L'aplicació web està comunicada seguint el següent diagrama:

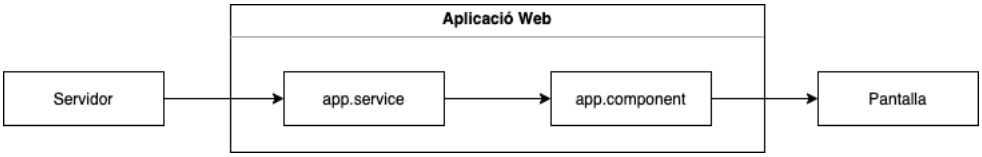


Fig. 4.4: Diagrama de l'aplicació web

El fitxer *app.service*, crida el servidor a la recerca de missatges. Concretament, crida al mòdul de *socket*. El component es subscriu a la funció del servei, de tal manera, que cada vegada que el servei rebi nova informació, aquesta passarà directament al component, el qual ho mostrarà per pantalla.

Si la informació que es rep, és una notificació conforme s'ha començat a gestionar un tiquet, el component mostrarà una barra de càrrega. Altrament, si el missatge que rep és un URL, el component mostrarà un un missatge per tal que el client pugui rebre el tiquet.

5 Tecnologies Utilitzades

Per tal de satisfer els requisits i l'arquitectura que es mostren els apartats anteriors, s'han seleccionat aquestes tecnologies.

5.1 Google Cloud

Per digitalitzar els tiquets es farà servir *Google Cloud*, *PaaS* (Platform as a service), el qual et proveeix de múltiples microserveis. Algunes tecnologies similars són *Amazon Web Service* o *Microsoft Azure*, però al final he utilitzat *Google Cloud*, ja que és gratuït i quan et registres et donen 300€ per gastar en els productes que vulguis.[10]

Fig. 5.1: Google Cloud logo

5.1.1 Google Drive API

Concretament, es farà servir l'API de *Google Drive* de *Google Cloud*. Aquesta API ens permet pujar fitxer en un directori concret de *Google Drive* a partir d'un programa. Hi ha altres serveis que em servien per fer el projecte (*AWS S3* per exemple), però la majoria són de pagament. L'API de *Google Drive* ens dóna fins a 1000000 crides gratuïtes, cosa que feia que s'abaratís el cost.[11]

Fig. 5.2: Google Drive logo

5.2 RaspberryPi 3

Una *RaspberryPi*, és un ordinador monoplaca que funciona amb el sistema operatiu Raspbian, una versió adaptada de Debian, i tot el seu codi és obert. Últimament, les *Raspberry* s'estan utilitzant molt en assumptes de domòtica, com per exemple convertir el televisor en un Smart TV o crear el teu propi sistema de música en streaming. En aquest cas faré servir la *RaspberryPi 3* com una impressora, el que implica que es detectarà com a tal en tots els dispositius connectats al wifi, i els documents que es vulguin imprimir es guardaran en una carpeta dins de la Raspberry.

La funció de dispositiu, la pot fer qualsevol ordinador, però s'ha elegit la *RaspberryPi 3*, ja que és un dispositiu baix cost que ens permet satisfer tots els requisits especificats.

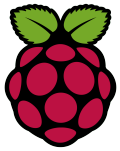


Fig. 5.3: RaspberryPi logo

5.2.1 Cups

CUPS significa Çommon UNIX Printing System", el qual ens permet configurar la RaspberryPi com un servidor d'impressió. A més, ens deixa instal·lar més impressores i compartir-les.

Fig. 5.4: CUPS logo

5.2.2 CUPS-PDF

CUPS-PDF és un software que configura una impressora dins del servidor d'impressions (en aquest cas CUPS). Quan la impressora configurada per CUPS-PDF rep una nova ordre, aquesta es converteix a PDF i es guarda en un directori específic de l'ordinador.[12]

5.3 Git

Git és un software de control de versions, que consisteix en guardar tots els canvis que s'han anat fent en el codi. Es farà servir per guardar tot el codi del servidor i l'aplicació web.

Fig. 5.5: Git logo

5.3.1 Github

Github és un servei de hosting de repositoris *git*. En el cas d'aquest projecte, s'utilitzarà per guardar el repositori *git* que conté tot el codi de l'aplicació.[13]

Fig. 5.6: GitHub logo

5.4 Python

Python és un llenguatge de programació, normalment utilitzat en Backend o en integració de sistemes. En aquest cas utilitzaré Python per programar un servidor que detecti tots els documents que arriben amb les ordres d'impressió a la *Raspberry*, firmar-les digitalment i finalment, pujar-les a *Google Drive*. S'ha triat *Python*, ja que és un llenguatge d'alt nivell i senzill d'utilitzar per a implementar aplicacions de backend.[14]

Fig. 5.7: Python logo

Per tal de desenvolupar el backend, faré ús dels softwares i llibreries de *Python* esmentats en els següents apartats:

5.4.1 PyCharm

Per programar Python, he fet servir *PyCharm*, un *IDE* (Integrated development environment) per implementar *Python*, desenvolupat per la companya de JetBrains. Conté anàlisis de codi, debugger, testos integrats, etc. S'utilitzarà aquest editor, ja que actualment, és dels més estesos a l'hora de programar amb *Python*.

Fig. 5.8: PyCharm logo

5.4.2 Flask

Flask és un framework de Python que ens permet crear aplicacions de forma ràpida. En aquest cas, l'utilitzaré per comunicar el servidor de *Python* amb una petita aplicació web feta amb d'*Angular*, i que serà la responsable de mostrar l'URL del tiquet una vegada aquest s'ha pujat a Google Drive.[15]

Fig. 5.9: Flask logo

5.4.3 Watchdog

Watchdog és una API de Python que serveix per monitoritzar els events del sistema de fitxers. S'utilitzarà per detectar tots els documents que arriven a la *RaspberryPi* i començar tot el procés per digitalitzar el tiquet.

5.4.4 Pydrive

PyDrive és una llibreria de *Python* que s'utilitza per simplificar moltes tasques a l'hora de tractar amb l'API de Google Drive. Faré servir aquesta llibreria per penjar tots els tiquets al núvol.

5.4.5 Endesive

Endesive és una llibreria de *Python* que s'utilitzarà per firmar els documents/tiquets digitalment abans de pujar-los al núvol.

5.5 Angular

Angular és un framework per aplicacions web de codi obert, que s'utilitza per crear i mantenir aplicacions web. Perquè el client pugui rebre el seu tiquet, faré una petita aplicació que mostri les URLs del tiquet en format codi QR. Hi ha altres llibreries/frameworks per desenvolupar aplicacions web, com ara *Vue* o *React*, però en ser una aplicació tan senzilla, *Angular* t'autogenera una gran part de codi, i per tant, els únics canvis que cal fer, són sobre el mateix component que es genera.[16]

Fig. 5.10: Angular logo

Per tal de desenvolupar l'aplicació web amb *Angular* faré ús dels següents softwares i llibreries:

5.5.1 Visual Studio Code

Visual Studio Code és un editor de codi, de codi obert, que s'utilitza per programar i debuggar pàgines web o aplicacions de cloud. Faré servir aquest editor per programar tota l'aplicació d'*Angular*. S'ha triat aquest editor de codi, ja que actualment, és l'editor més estès per al desenvolupament d'aplicacions web.

Fig. 5.11: Visual Studio Code logo

5.5.2 TypeScript

TypeScript és un llenguatge de programació tipat de codi obert. És un superconjunt de *Javascript*, el qual afegeix tipus i classes. Com que Angular funciona damunt de *TypeScript*, faré servir el llenguatge per programar l'aplicació web.[17]

Fig. 5.12: TypeScript logo

5.5.3 Node

Node, és un entorn de programació dissenyat per escriure aplicacions basades en *JavaScript*. Per tal que funcioni *Angular*, és necessària la seva instal·lació, ja que aquest funciona damunt de Node.[18]

Fig. 5.13: Node logo

5.5.4 Npm

Npm és un sistema de gestió de paquets de *JavaScript*. És necessari per instal·lar *Angular* i tots els paquets que s'utilitzaran per dur a terme l'aplicació web.[19]

Fig. 5.14: Npm logo

5.5.5 @techiediaries/ngx-qrcode

ngx-qrcode és una llibreria d'*Angular* de codi obert, que serveix per generar codis QR de forma ràpida. La faré servir per mostrar l'URL, en format de codi QR, una vegada la rebem del servidor de *Python*.

5.6 Socket IO

Socket IO és una llibreria de codi obert que s'utilitza per mantenir una connexió persistent un servidor i un o múltiples clients. D'aquesta manera, si un client o el servidor envien un missatge per aquesta connexió, el rebran totes les connexions a la vegada. En el cas d'aquest projecte, utilitzaré la llibreria de *Socket IO* per mantenir el client informat sempre que s'hagi digitalitzat un nou tiquet.[20]

Fig. 5.15: Socket IO logo

5.6.1 ngx-socket-io

Llibreria d'Angular que integra les funcionalitats de *Socket IO*.

5.6.2 flask-socketio

Llibreria de Python que integra les funcionalitats de *Socket IO*, sempre que hi hagi Flask involucrat.

5.7 Docker

Docker és un projecte de codi obert que permet automatitzar el desplegament de les aplicacions dins de quasi qualsevol sistema operatiu. Això em permetrà que cada vegada que s'encengui la *RaspberryPi*, tant l'aplicació de Python (encarregada de digitalitzar el tiquet) i l'aplicació d'*Angular* (encarregada de mostrar l'URL del tiquet en format de codi QR), comencin a funcionar automàticament.[21]

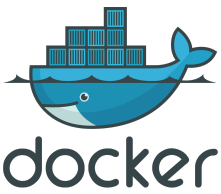


Fig. 5.16: Docker logo

6 Implementació

En aquest capítol, s'expliquen tots els passos que he seguit per tal de tirar el projecte endavant, des de la configuració de la RaspberryPi a la configuració del Docker.

Aclarir que tota la implementació del projecte s'ha dut a terme fora de la RaspberryPi, i s'ha instal·lat dins de la Raspberry posteriorment.

6.1 Configuració de la RaspberryPi 3

El primer pas per fer el projecte, és configurar la RaspberryPi 3 perquè es pugui detectar com una impressora dins de la xarxa wifi. Per fer-ho, en primer lloc he hagut de configurar el sistema operatiu de la RaspberryPi, i finalment configurar CUPS.

6.1.1 Sistema operatiu

Primerament, per muntar el sistema operatiu a la RaspberryPi, es necessita una targeta SD, la qual és la que conté tota la memòria de la Raspberry mentre està funcionant. A partir d'aquí, els passos que he seguit són els següents:

Descarregar Raspberry Pi Imager

Raspberry Pi Imager, és un software de Raspberrry Pi Fundation que permet instal·lar fàcilment sistemes operatius a la targeta SD, i que aquest funcioni correctament a la Raspberry.

Per fer-ho, només cal anar a la següent URL (**https://www.raspberrypi.org/software/**), descarregar-lo directament des del teu PC, i finalment instal·lar-lo.

Formatejar la targeta SD

Per continuar, s'ha d'inserir la targeta SD al PC. Una vegada això està fet, s'ha d'executar el software Raspberry Pi Imager.

Una vegada executat, apareix la següent pantalla:

Fig. 6.1: Raspberry Pi Imager: Pàgina d'inici

El primer que s'ha de fer és seleccionar el sistema operatiu. Per fer-ho, triem l'opció de *CHOOSE OS*, i en la pantalla següent, seleccionem la primera opció (Raspberry Pi OS (32-bit)).

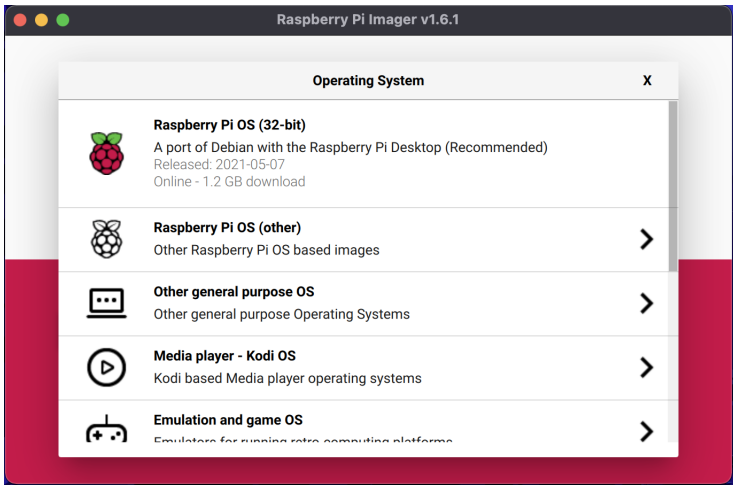


Fig. 6.2: Raspberry Pi Imager: Selecció de sistema operatiu

Una vegada s'ha seleccionat el sistema operatiu, el mateix software torna a la pàgina principal. Una vegada allí, s'ha de seleccionar la targeta SD on es vol instal·lar el sistema operatiu. Per fer-ho, seleccionem l'opció de *CHOOSE STORAGE*, i seguidament, seleccionem el dispositiu de memòria.

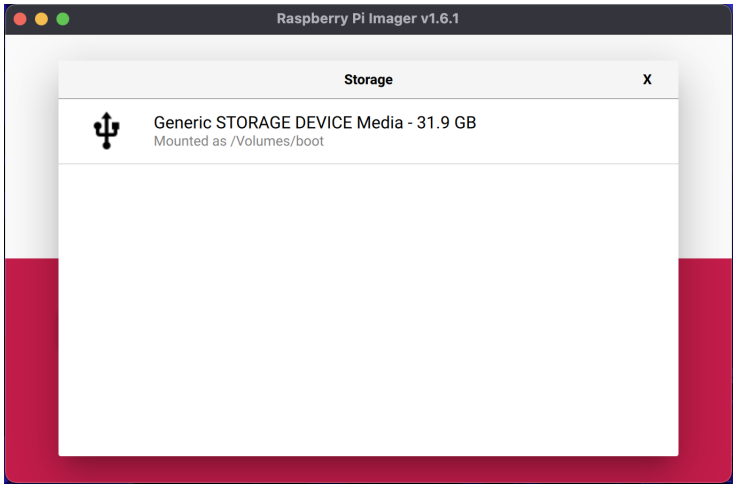


Fig. 6.3: Raspberry Pi Imager: Selecció de targeta SD

Seguidament, només cal clicar l'opció de *WRITE* de la pàgina principal (el qual s'habilita una vegada s'han fet els dos passos anteriors).

Fig. 6.4: Raspberry Pi Imager: Instal·lant el sistema operatiu

Finalment, quan s'ha instal·lat el sistema operatiu en la targeta SD, apareix un missatge assegurant que s'ha instal·lat el sistema operatiu, i que la targeta SD ja es pot extreure.

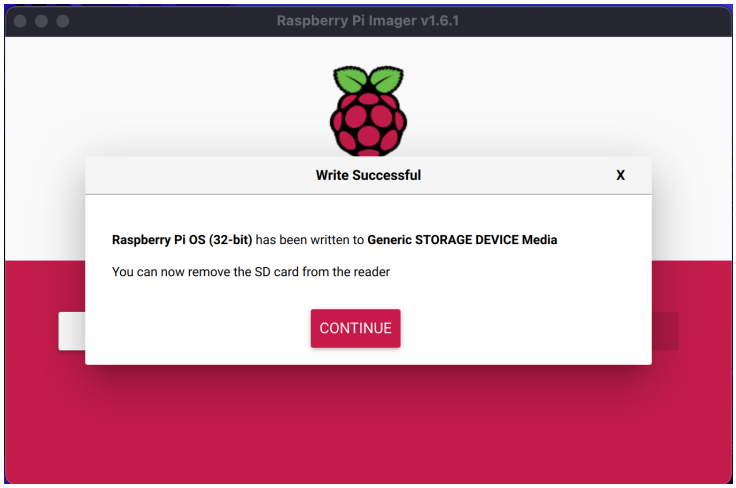


Fig. 6.5: Raspberry Pi Imager: Missatge de finalització

Configuració del sistema operatiu

Quan ja tenim el sistema operatiu instal·lat a la targeta SD, aquesta s'ha d'inserir a la RaspberryPi, i aquesta s'endolla a una presa de corrent. Una vegada s'ha endollat, aquesta s'encendrà automàticament, i començarà el procés de configuració del SO.

L'únic que s'ha de configurar és el següent:

- Idioma i regió.
- Contrasenya de l'usuari (per als següents passos, consideraré l'usuari pi i la contrasenya 1234).
- Configuració de la pantalla.
- Configuració de la wifi.
- Actualitzar el sistema operatiu en cas que sigui necessari.

Quan ja tenim tot el sistema configurat, ja podem començar a configurar la Raspberry com a impressora. Per fer-ho, s'utilitzarà CUPS.

6.1.2 Configuració de CUPS

Per configurar CUPS, i fer que la RaspberryPi es detecti com a impressora dins de la xarxa wifi, he procedit amb els següents passos:

Instal·lar CUPS

Primer que res s'ha d'obrir una nova finestra de terminal, i executar les següents comandes:

```
$ sudo apt update
$ sudo apt-get install cups
```

Aquestes dues línies de codi, instal·len CUPS a la RaspberryPi. La primera comanda actualitza la informació dels paquets configurats al sistema. La segona, instal·la CUPS. En totes dues comandes s'executa *sudo* anteriorment, el qual fa que les comandes s'executin amb drets d'administració. En cas que demani la contrasenya, s'ha de ficar la contrasenya pertinent a l'usuari amb privilegis d'administració (En aquest cas, com s'ha dit abans, 1234).

Afegir pi com a usuari admin

Seguidament, s'afegeix a l'usuari pi dins del grup d'administració amb la següent comanda:

```
$ sudo usermod -a -G lpadmin pi
```

Obrir CUPS

Amb les comandes del primer pas s'ha instal·lat CUPS i ja està funcionat, però de moment no es pot accedir de forma remota. Per fer que CUPS es pugui cridar des de qualsevol dispositiu, s'executa les següents comandes:

```
$ sudo cupsctl --remote-any
$ sudo /etc/init.d/cups restart
```

Instal·lar cups-pdf

Com que en un futur es voldrà guardar en format PDF totes les comandes d'impressió que arribin a la RaspberryPi, s'instal·larà cups-pdf. Aquest paquet agafa els fitxers que arriben a CUPS, i els converteix a PDF. Per fer-ho, s'executa la següent comanda:

```
$ sudo apt-get install cups-pdf
```

Configurar CUPS

Una vegada s'ha instal·lat tots els paquets necessaris per fer funcionar la Raspberry de la manera desitjada, toca configurar CUPS.

En primer lloc, amb el navegador s'ha d'accedir a la direcció de CUPS a través del navegador. Per saber quina és la IP assignada que té la Raspberry, s'executa la següent comanda:

```
$ ifconfig
```

La comanda de *ifconfig* ens dóna molta informació sobre les interfícies de xarxa del nostre PC. Per trobar quina és la IP assignada a la RaspberryPi, ens hem de fixar en *wlan0*, i seguidament, a la segona línia, hem d'agafar el valor de *inet*. Es pot veure un exemple en la següent imatge:

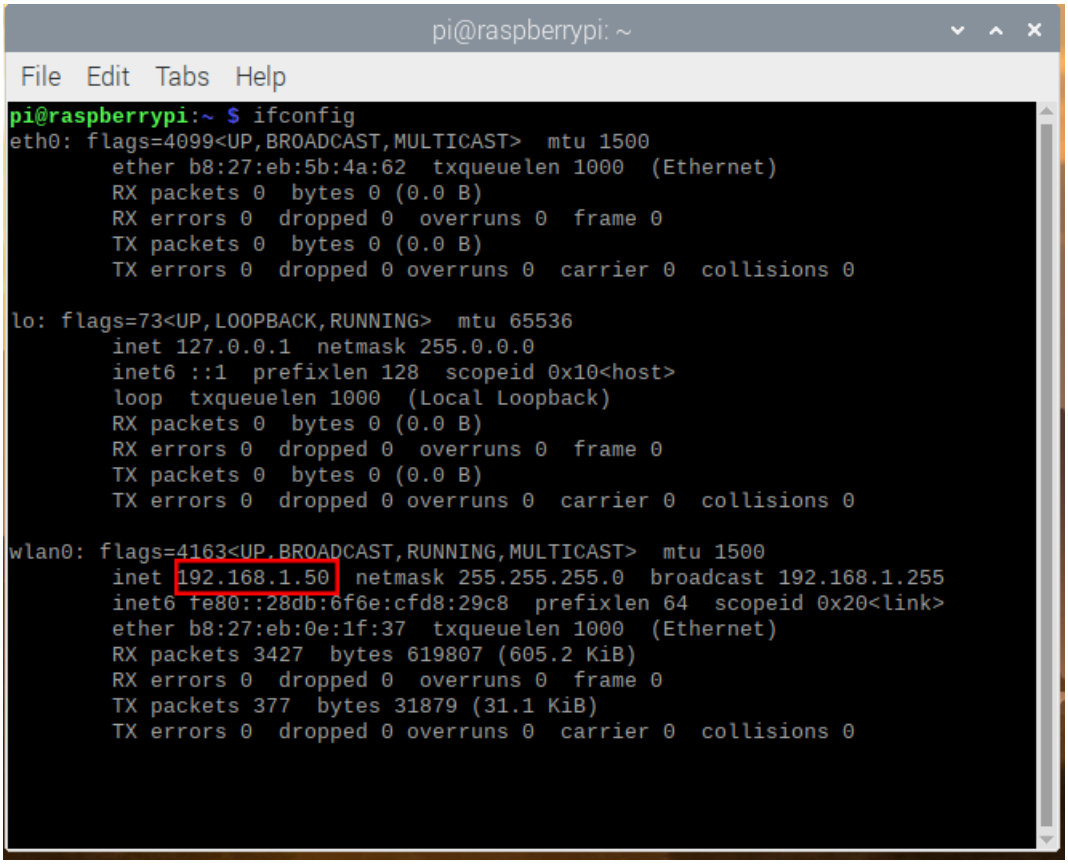


Fig. 6.6: Execució de la comanda ifconfig

Una vegada tenim la IP, obrim una pestanya del navegador, escrivim la IP, i seguidament el port 631. Seguint l'exemple de la imatge anterior, quedaria una direcció de la següent forma: *192.168.1.50:631*.

La primera pàgina que s'ha de veure una vegada entreu a l'adreça IP amb port 631, és la següent:

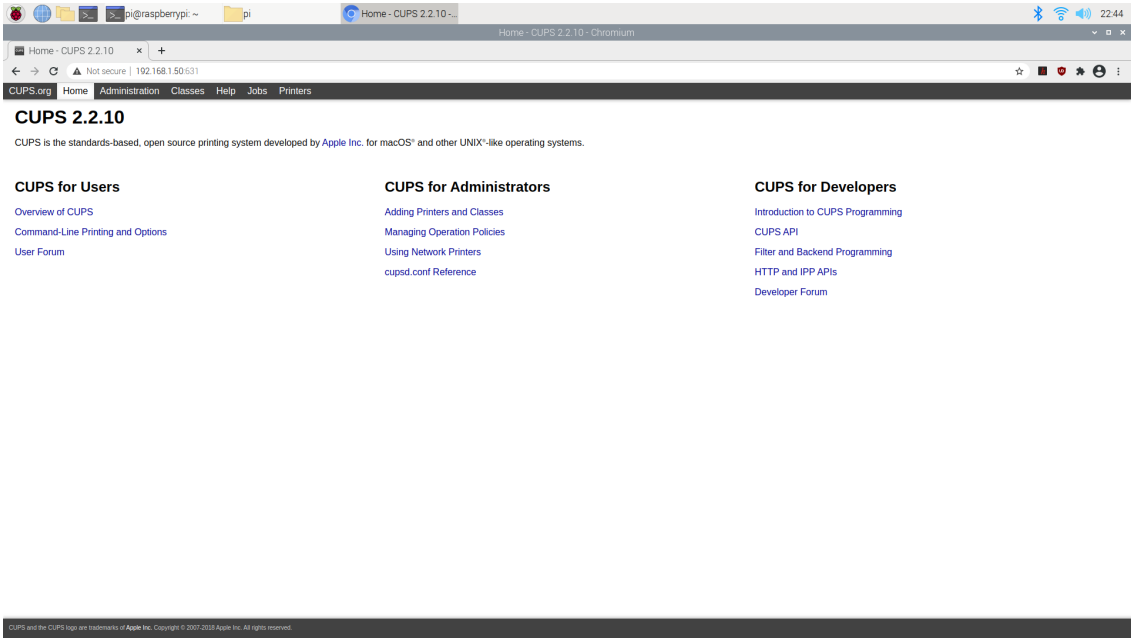


Fig. 6.7: CUPS: Pàgina d'inici

Des de la pàgina d'inici de CUPS, s'ha de fer clic a la pestanya de *Printers* que es troba en la part superior dreta.

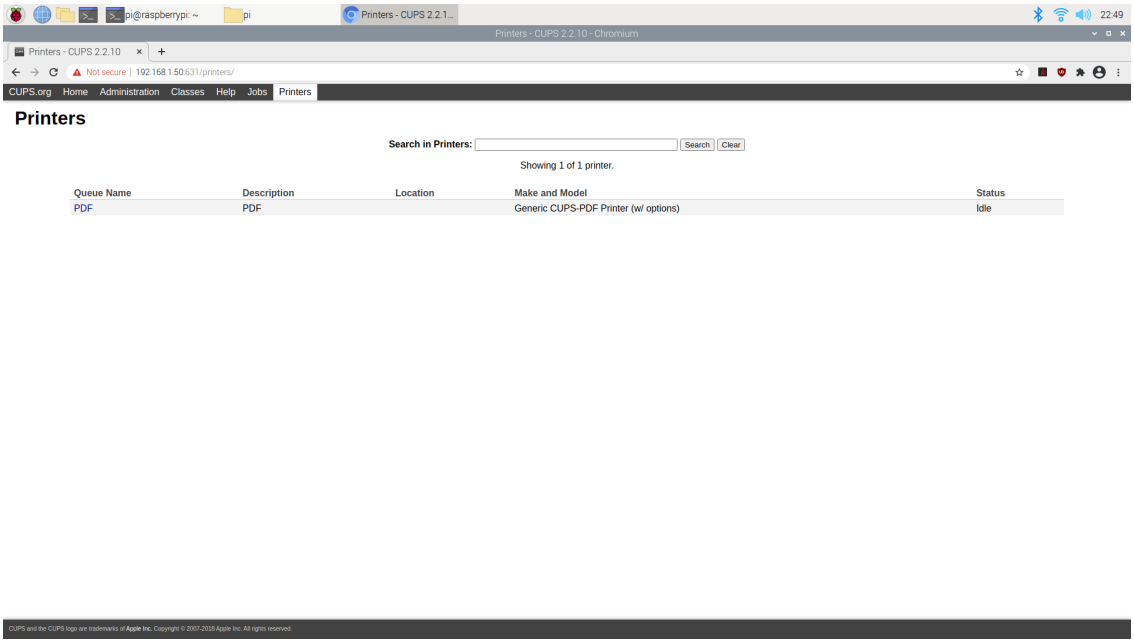


Fig. 6.8: CUPS: Pàgina Printers

Seguidament, només cal fer clic a l'única impressora que tenim configurada, PDF (la qual s'afegeix automàticament en instal·lar cups-pdf). Si s'han seguit els passos, s'hauria de mostrar una pantalla similar a la següent:

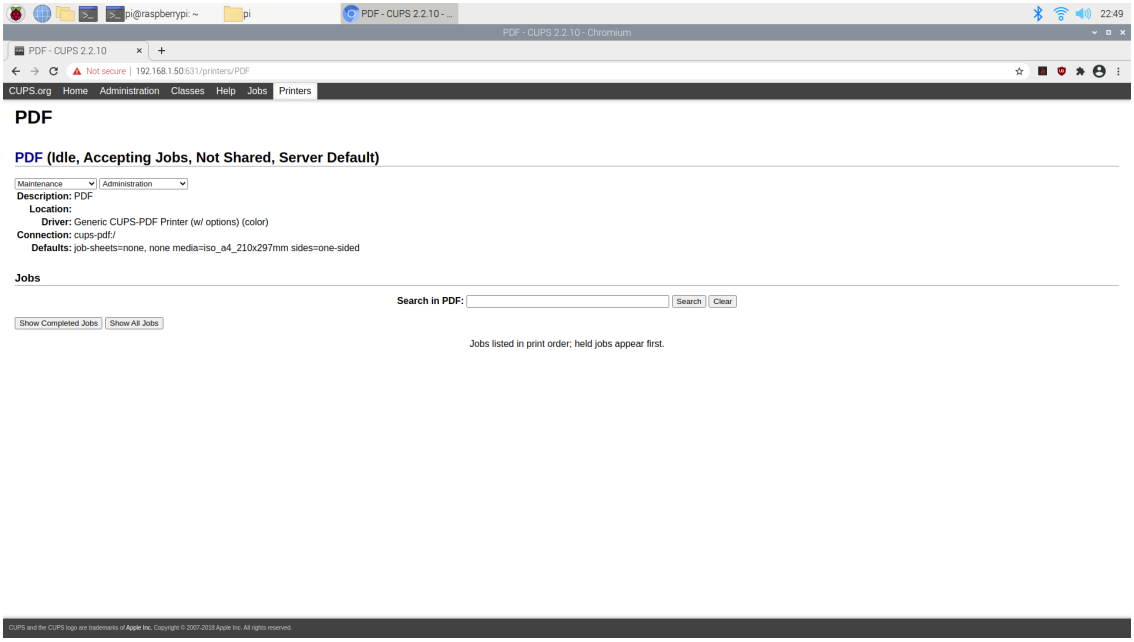


Fig. 6.9: CUPS: Pàgina Impressora PDF

Com que la impressora ja està instal·lada, només cal modificar la seva configuració per tal que aquesta sigui accessible. Per fer-ho, s'ha de fer clic al desplegable on posa *Administration*, i seguidament clicar *Modify Printer*. S'hauria de mostrar la següent pantalla.

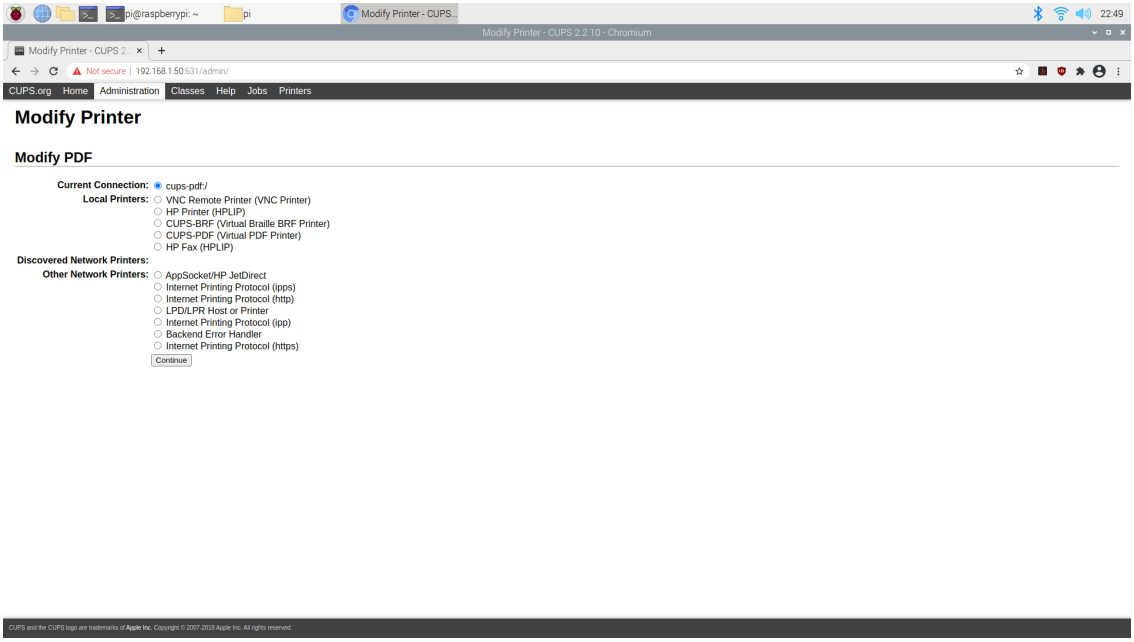


Fig. 6.10: CUPS: Pàgina de modificació de la impressora 1

La modificació de la impressora consisteix en 3 pantalles diferents, però només cal que ens centrem en la segona, la qual ens demana el nom amb el qual volem mostrar la impressora a través de la xarxa wifi, i si la volem compartir (el qual s'ha d'activar si o si, per tal que es pugui detectar a través de la xarxa). Tal com es fa pot veure en la següent captura, en aquest cas li he ficat el nom de *RaspberryPrinter*, i se li ha de donar la marca de *Share This Printer*.

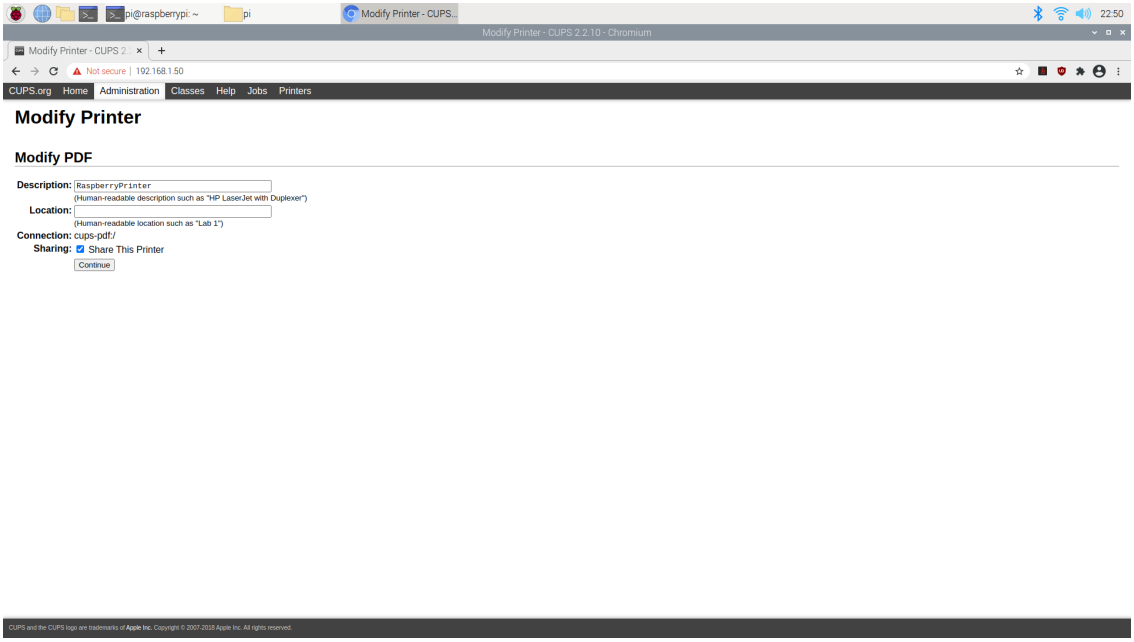


Fig. 6.11: CUPS: Pàgina de modificació de la impressora 2

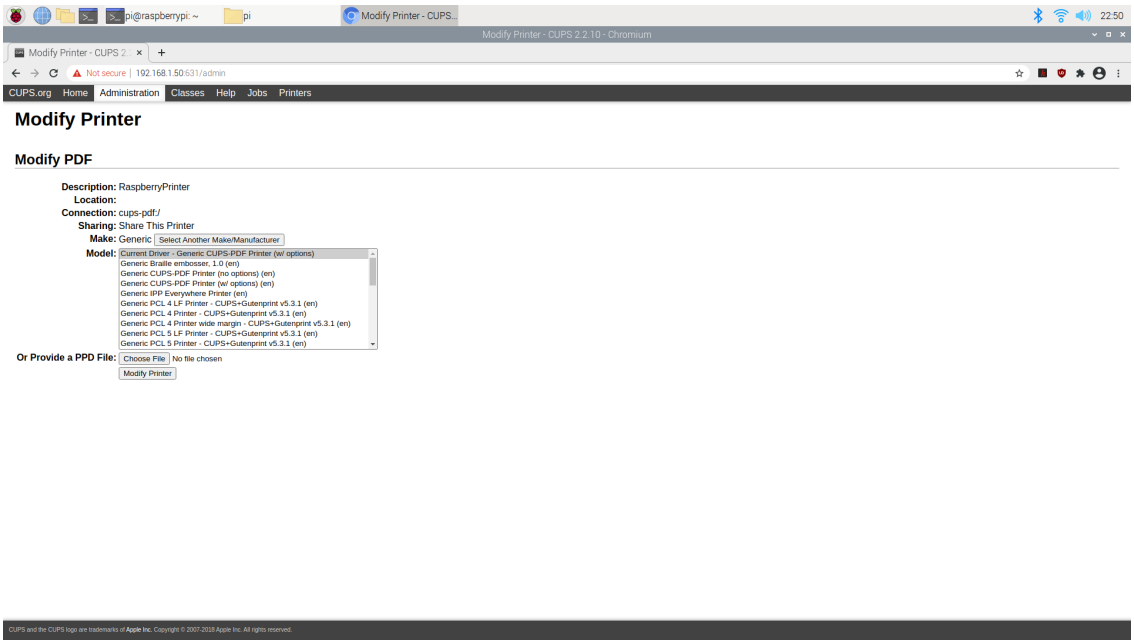


Fig. 6.12: CUPS: Pàgina de modificació de la impressora 3

Una vegada s'han completat tots els passos anteriors, es mostra el següent missatge, confirmant que la impressora s'ha modificat correctament.

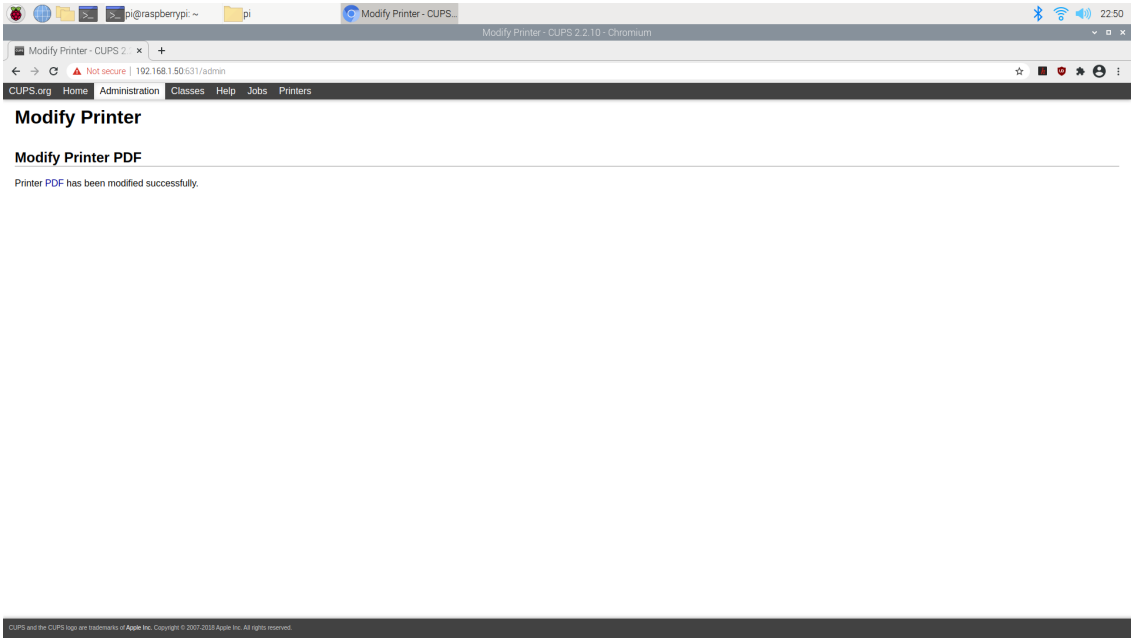


Fig. 6.13: CUPS: Pàgina d'inici 2

Finalment, s'ha d'executar la següent comanda a la terminal de la RaspberryPi per tal de fer els canvis efectius:

```
$ sudo /etc/init.d/cups restart
```

Arribats a aquest punt, ja podem detectar la impressora dins de la xarxa wifi, i podem enviar documents a imprimir. Tots els documents que s'enviïn a imprimir a la RaspberryPi, es guardaran en format PDF al directori */var/spool/cups-pdf/ANONYMOUS*, el qual s'haurà de tenir en compte a l'hora d'implementar el servidor.

6.2 Github

Abans de començar a implementar el servidor, es crea un repositori *git* a *Github*, el qual ens servirà per poder mantenir tot l'històric del codi.

Per fer-ho, es seguiràn els següents passos:

1. Entrar al a pàgina de GitHub: **Github**

2. Inicia la sessió, o en cas que no tinguis compte, registrar-te.

3. Anar al l'opció de *Your repositories* en el desplegable que es troba a la part de dalt a la dreta de la pàgina.

4. Fer clic a l'opció de *New*

5. Omplir els camps que es demanen (Repository Name, Public / Private, etc...)

6. Donar clic a l'opció de *Create repository*

Quan ja s'hagin dut a terme els passos anteriors, apareixerà una pàgina que ens dóna la informació necessària per sincronitzar el repositori de *git* que s'acaba de crear. En el cas que no tinguis *git* instal·lat, instal·la'l.

Per començar a desenvolupar el servidor, s'obrirà una terminal, s'anirà fins al directori on volem guardar tot el projecte, i finalment es clonarà el repositori seguint la següent comanda:

```
$ git clone https://github.com/{username}/{repository_name}.git
```

Finalment, s'entra al directori amb nom *repository_name*, i es crearan dues carpetes més: *api* i *web*. La carpeta *api*, servirà per guardar tot el projecte que conté el servidor. La carpeta *web*, contindrà l'aplicació web que mostrarà el codi QR. Finalment, a l'arrel del directori principal, hi haurà tota la configuració de *Docker*. L'estructura quedaria d'aquesta forma:

```
repository_name
├── api/
├── web/
└── docker-config
```

6.3 Implementació del servidor

El següent pas del projecte, és crear el servidor que es farà servir per gestionar els tiquets. Primerament, es configurarà *Flask* per tal de converir el nostre servidor en una API. Seguidament, es configurarà *SocketIO* per tal de poder enviar l'estat dels tiquets a l'aplicació web. Es continuarà amb la implementació de *Watchdog*, per detectar totes les comandes d'impressió. Seguidament, es crearà un certificat digital, i s'implementarà la firma de documents. Finalment, es configurarà *Google Cloud* i s'implementarà la pujada de fitxers a *Google Drive* mitjançant *pydrive*

6.3.1 Flask i requeriments

El primer que es farà, és obrir el directori *api/* des de *PyCharm*. Una vegada s'hagi fet això, es crearan 2 fitxers (*.gitignore* i *requeriments.txt*) i el mòdul *wetick*.

.gitignore

En aquest fitxer, s'afegeix un llistat de fitxers i directoris els quals volem que no es sincronitzin amb repositori de *git*.

requeriments.txt

En aquest fitxer, s'especifiquen totes les llibreries externes que es volen utilitzar. En el cas d'aquest projecte, s'han d'afegir les següents:

```
flask
flask-cors
flask-socketio
watchdog
pydrive
cryptography
endesive
```

Una vegada ja s'han afegit totes les llibreries, s'executarà *pip* amb la línea de comandes:

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

pip, és un sistema de gestió de paquets de *Python*. S'ha d'instal·lar en cas que no ho estigui.

Per mantenir les versions dels paquets que s'han instalat a l'executar la comanda, i assegurar-nos que sempre estem treballant amb els mateixos, s'executa la següent comanda:

```
$ pip3 freeze -r requirements.txt > requirements.lock
```

Aquesta comanda, crea un nou fitxer (*requeriments.lock*), on s'especifiquen els paquets i versions que s'han instal·lat. A més, també especifica totes les dependències d'aquests.

Mòdul wetick

Per fer que el servidor estigui funcionat (tot i que no faci cap acció en aquest moment), es procedirà amb els següents passos:

1. Crear el fitxer *__main__.py*

2. Crear el fitxer *wetick.py*

3. Dins del fitxer de *wetick.py*, s'escriuràn les següents línies:

```
from flask import
from flask_cors import CORS

app = Flask(__name__)
CORS(app)
```

4. Dinc del fitxer *__main__.py*, s'escriuràn les següents línies:

```
from wetick.wetick import app

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Quan ja es tenen els fitxers escrits, en el moment que s'executa l'aplicació, aquesta comença a córrer a **localhost:5000**. Finalment, l'estructura de carpetes que s'hauria de tenir en aquest punt en el directori *api/* és el següents:

```
api/
├── requeriments.txt
├── requeriments.lock
├── .gitignore
└── wetick
    ├── __main__.py
    └── __wetick__.py
```

6.3.2 SocketIO

Ara mateix tenim un servidor funcionat, però el qual no realitza cap acció. Per tant, el següent pas serà implementar el *socket* que permetrà notificar l'estat dels tiquets a l'aplicació web. Per fer-ho, es seguiran els següents passos:

Mòdul enums

En primer lloc, dins del mòdul *wetick*, es crearà un altre mòdul anomenat *enums*. Dins d'aquest mòdul, es crearà un fitxer (*status_enums.py*), que conté els possibles estats del tiquet: *NoTicket*, *TicketProcessing* i *TicketProcessed*. El contingut d'aquest fitxer ha de ser el següent:

```
from enum import Enum

class Status(Enum):
    NoTicket = 1
    TicketProcessing = 2
    TicketProcessed = 3
```

Modificació wetick i __main__

Com que es vol que el servidor executi el *socket*, no una API REST, es modificarà el fitxer *wetick.py* de la següent manera:

```
from flask import
from flask_cors import CORS
from flask_socketio import SocketIO

app = Flask(__name__)
socket_io = SocketIO(app, cors_allowed_origins="*")
CORS(app)
```

El fitxer *__main__* també es modifica. Ha de quedar amb el següent contingut:

```
from wetick.socket import socket_actions

if __name__ == '__main__':
    socket_actions.run_socket()
```

Tot i que això no funciona, s'arregla en la següent secció, ja que s'implementa el funcionament del socket.

Mòdul socket

Seguidament, dins del mòdul *wetick*, es crearà un altre mòdul anomenat *socket*. Dins d'aquest mòdul, es crearà el fitxer *socket_actions.py*, el qual contindrà totes les accions del socket. Seguidament, s'afegirà el següent codi dins del fitxer *socket_actions.py*:

```
from wetick.enums.status_enum import Status
from wetick.wetick import app, socket_io

@socket_io.on('message')
def manage_ticket(url=None):
    if url:
        socket_io.emit(
            'message',
            dict(status=Status.TicketProcessed.name, url=url)
        )
    else:
        socket_io.emit(
            'message',
            dict(status=Status.TicketProcessing.name)
        )

def run_socket():
    socket_io.run(app)
```

Aquest codi fa que quan el servidor s'executa, el *socket* comenci a funcionar automàticament (per defecte al port 5000). Addicionalment, també es poden afegir funcions amb capçals com *@socket_io.on('connect')* (que permet saber quan un nou client s'ha connectat al *socket*).

6.3.3 Watchdog

Ara que el *socket* ja està funcionant, s'ha d'afegir tots la lògica per detectar els documents. Per fer-ho, s'utilitzarà la llibreria *watchdog*, i es seguirà els següents passos:

Mòdul file_listener

Primerament, dins del mòdul *wetick*, es crearà un altre mòdul anomenat *file_listener*. Dins d'aquest mòdul, es crearà un fitxer (listener.py), el qual serà l'encarregat d'escoltar tots els fitxers PDF que arrivin al directori */var/spool/cups-pdf/ANONYMOUS*. Dins d'aquest fitxer, afegirem el següent contingut:

```
import os
from watchdog.observers import Observer
from watchdog.events import PatternMatchingEventHandler
from wetick.socket import socket_actions

def _on_created(event):
    print(f"hey, {event.src_path} has been created!")
    socket_actions.manage_ticket()

def listen():
    patterns = ["*"]
    ignore_patterns = None
    ignore_directories = False
    case_sensitive = True
    my_event_handler = PatternMatchingEventHandler(
        patterns, ignore_patterns, ignore_directories, case_sensitive
    )
    my_event_handler.on_created = _on_created
    listen_path = "/var/spool/cups-pdf/ANONYMOUS"
    go_recursively = True
    my_observer = Observer()
    my_observer.schedule(
        my_event_handler, listen_path, recursive=go_recursively
    )
    my_observer.start()
```

La funció de *listen*, defineix el directori que es vol mirar i quines accions es volen dur a terme. En el meu cas, només vull capturar totes les accions de creació de nous fitxers (*_on_create*) al directori */var/spool/cups-pdf/ANONYMOUS*. La funció que he definit per gestionar el document que s'ha creat, crida la funció de *manage_ticket*, que s'ha creat en l'apartat anterior.

Modificació de socket_actions.py

Tot i això, el mòdul de *file_listener* no es crida des de cap lloc. Per fer que s'executi, s'ha de modificar el fitxer de *socket_actions.py*. Per una banda, s'ha d'importar el mòdul que s'acaba de crear:

```
from wetick.file_listener import listener
```

I seguidament, afegir la següent línia com la primera de la funció *run_socket()*:

```
listener.listen()
```

Per tant, ara mateix, si s'executa el programa, cada vegada que es crea un fitxer en el directori */var/spool/cups-pdf/ANONYMOUS*, el *socket* enviarà un missatge, i tots els clients que hi estiguin connectats el rebran. En aquest cas, només s'envia el missatge de *TicketProcessing* definit anteriorment en el fitxer *status_enums.py*

6.3.4 Certificació dels tiquets

Arribats a aquest punt, només cal gestionar el tiquet (signant-lo amb un certificat) i pujar-lo a *Google Drive*.

Crear certificat

Per crear el certificat digital, s'utilitzarà el programa *Keychain Access*, que ve per defecte en els sistemes operatius *MAC OS*.

En primer lloc, s'obrirà el programa *Keychain Access*, el qual hauria de mostrar una pantalla com la següent:

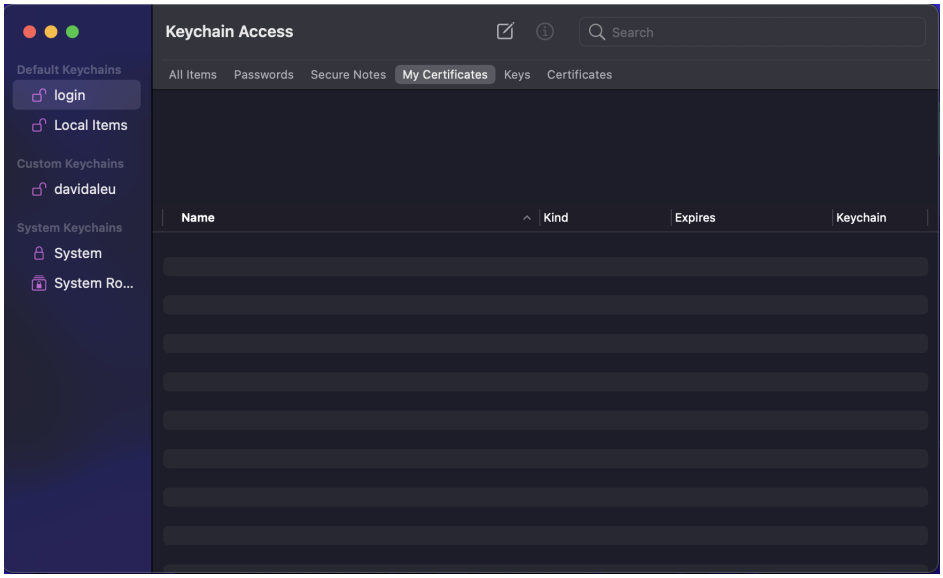


Fig. 6.14: Keychain Access: Pantalla d'inici

Des d'aquesta pantalla, s'ha de clicar *Keychain Access* a la barra de menú de la part superior, movem el cursor damunt de l'opció *Certificate Assistant* i finalment es clica *Create a Certificate...*. S'hauria de mostrar el següent:

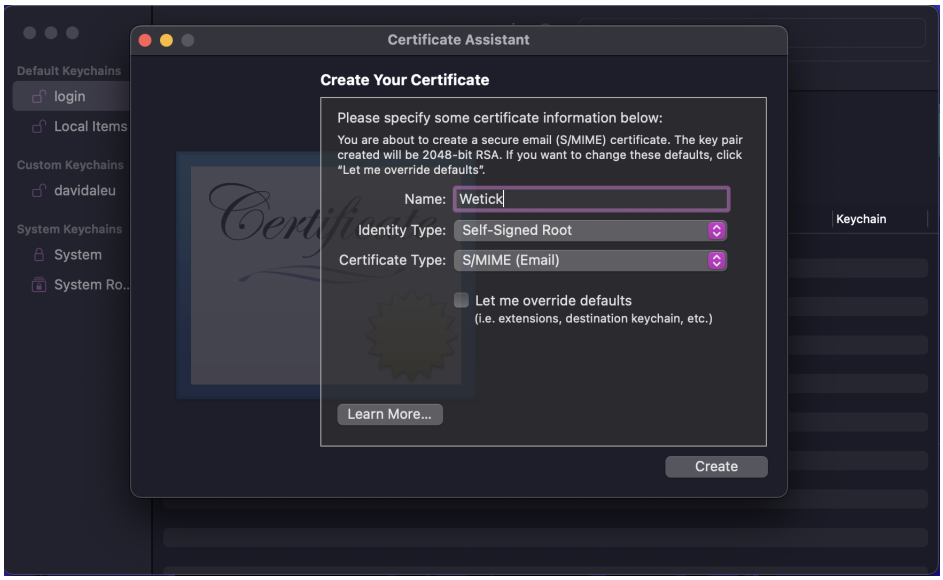


Fig. 6.15: Keychain Access: Crea certificat

En aquesta pantalla, s'ha d'omplir el nom del certificat, i seguidament, s'ha de donar clic al botó de *Create*. Seguidament, es torna a la pantalla d'inici, però es pot veure que el certificat s'ha guardat correctament.

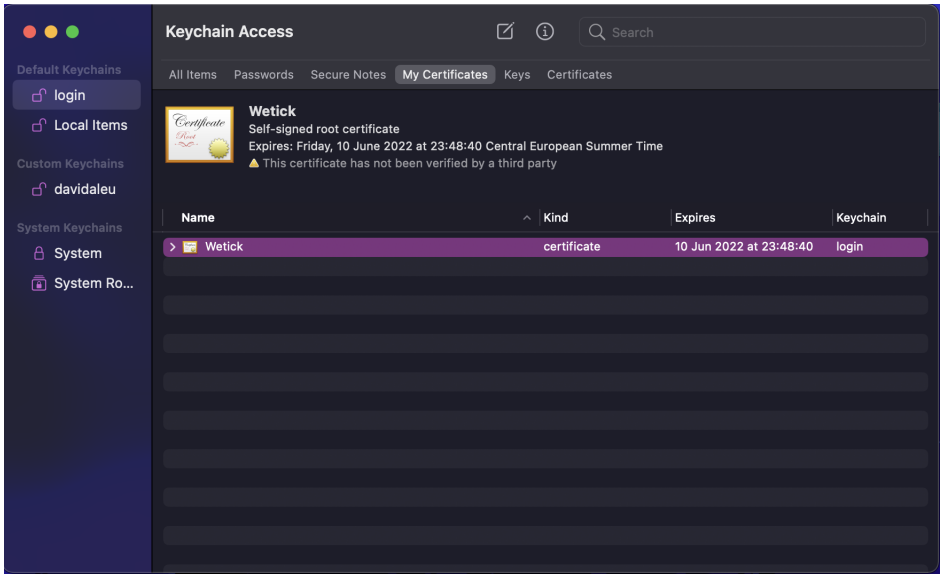


Fig. 6.16: Keychain Access: Pantalla d'inici 2

Com que més endavant s'ha d'utilitzar, s'exportarà el certificat. Per fer-ho, s'ha de donar clic amb el botó dret damunt del certificat que s'acaba de crear, i seguidament clicar *Export*. S'hauria de mostrar una pantalla com la següent:

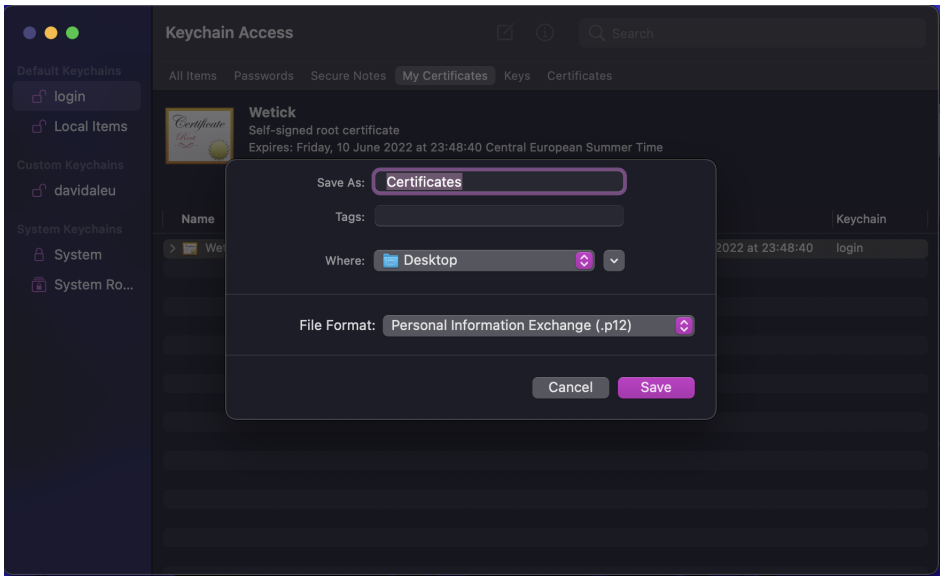


Fig. 6.17: Keychain Access: Exportar certificat 1

Es busca un lloc per guardar-ho, ja que després s'utilitzarà el fitxer que es genera, i seguidament, es dóna clic a *Save*.

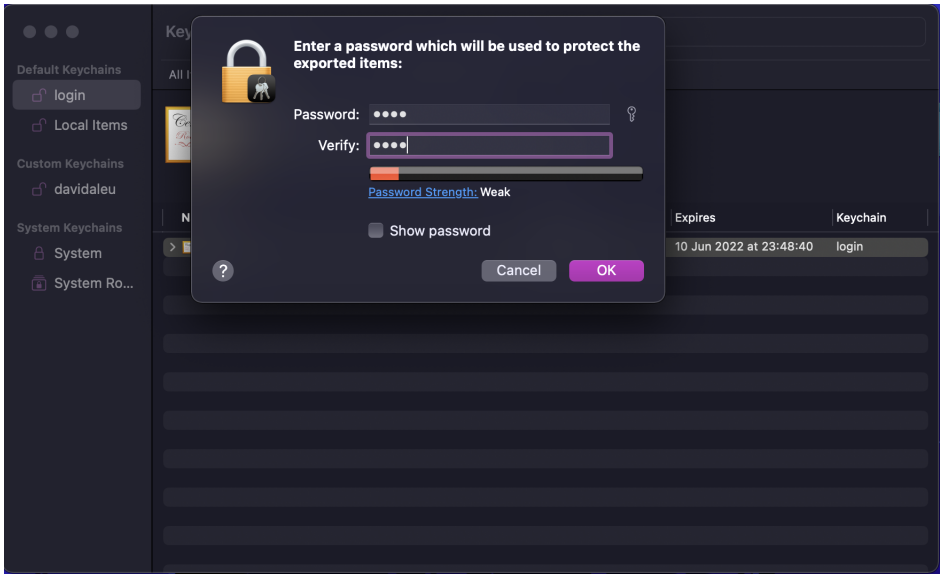


Fig. 6.18: Keychain Access: Exportar certificat 2

Quan es dugui a terme l'acció anterior, sortiran dos diàlegs. El primer demana una contrasenya per al certificat, i el segon demana la contrasenya de l'usuari que està amb la sessió iniciada, ja que és una acció que només poden fer els usuaris amb permisos d'administració.

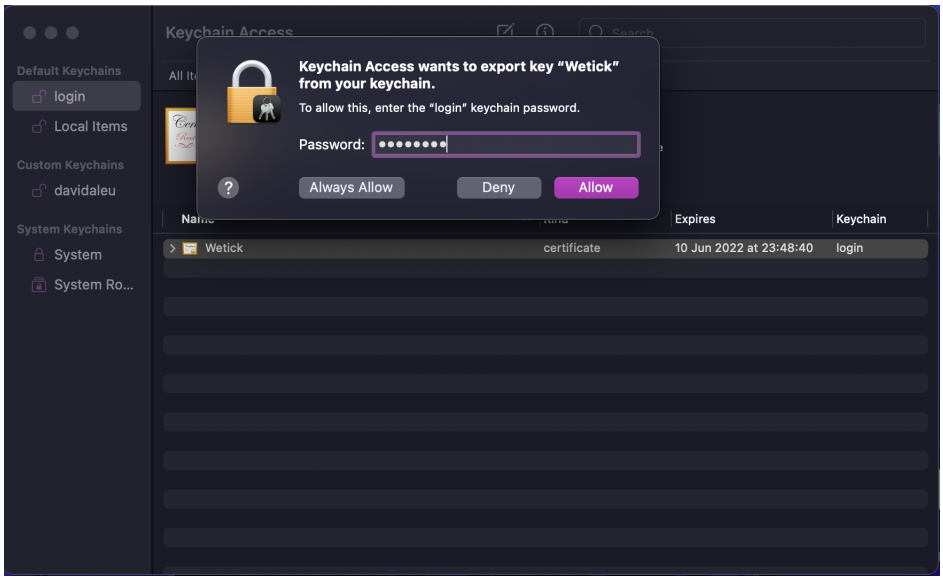


Fig. 6.19: Keychain Access: Exportar certificat 3

Finalment, ara que ja s'ha creat el certificat correctament, ja es pot implementar tota la lògica per firmar els fitxers PDF.

Mòdul file_sign

Primerament, dins del mòdul *wetick*, es crearà un altre mòdul anomenat *file_signer*. Dins d'aquest mòdul, es crearà un fitxer (signer.py), el qual serà l'encarregat de firmar tots els documents PDF. El contingut d'aquest fitxer serà el següent:

```
import datetime

from cryptography.hazmat import backends
from cryptography.hazmat.primitives.serialization import pkcs12

def sign(file_path):
    date = datetime.datetime.utcnow()
    date = date.strftime("D:%Y%m%d%H%M%S+00'00'")
    dct = {
        "aligned": 0,
        "sigflags": 3,
        "sigflagsft": 132,
        "sigpage": 0,
        "sigbutton": True,
        "sigfield": "wetick_sign",
        "auto_sigfield": True,
        "sigandcertify": True,
        "signaturebox": (470, 840, 570, 640),
        "signature": "Signature",
        "contact": "wetick@gmail.com",
        "location": "Barcelona",
        "signingdate": date,
        "reason": "Reason",
        "password": "1234",
    }
    with open("wetick/file_signer/WeTick.p12", "rb") as fp:
        p12 = pkcs12.load_key_and_certificates(
            fp.read(), b"1234", backends.default_backend()
        )

    datau = open(file_path, "rb").read()
    datas = cms.sign(datau, dct, p12[0], p12[1], p12[2], "sha256")
    new_path = file_path.split('/')[-1]
    with open(new_path, 'wb') as f:
        fp.write(datau)
```

```
            fp.write(datas)
        return new_path
```

Finalment, s'ha de modificar el fitxer *listener.py*. Per una banda, s'ha d'importar el mòdul que s'acaba de crear:

```
from wetick.file_sign import signer
```

I seguidament, afegir la següent línia com la tercera de la funció *_on_create()*:

```
file_path = signer.sign(event.src_path)
```

Amb aquest codi, es llegeix el PDF i es firma amb el certificat que s'ha creat al pas anterior. Per tal que funcioni, s'ha de moure el certificat dins del mòdul *file_signer*.

6.3.5 Configuració de Google Cloud

El següent pas és pujar els tiquets a *Google Drive*, però abans s'ha de configurar *Google Cloud* per tal que ens deixi fer crides remotes. Per fer-ho es seguiran els següents passos:

Habilitar Google Drive API

Primer que res, si no es té un compte de Google, s'ha de crear. Seguidament, s'ha d'accedir a la pàgina de **Google Cloud**. La pàgina que apareix a l'accedir-hi, és la següent:

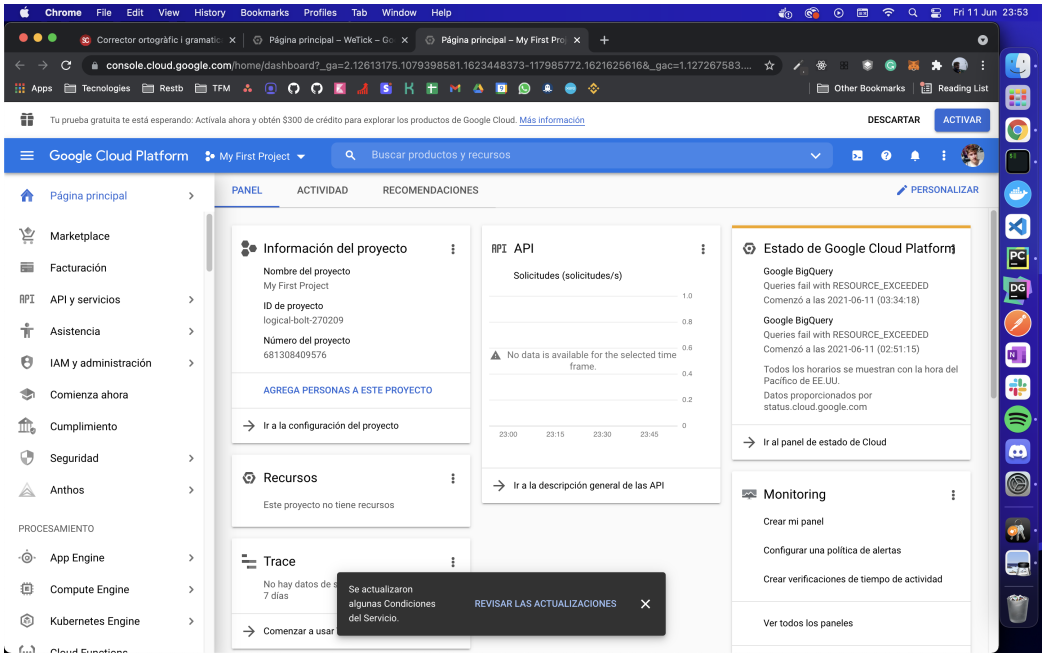


Fig. 6.20: Google Cloud: Pàgina d'inici

Si mai has utilitzat *Google Cloud*, La pàgina d'inici mostra tots els detalls del projecte *My First Project*, el qual ve creat per defecte. Per tal de tenir totes les coses del projecte unificades (en el cas que es vulguin afegir noves funcionalitats en un futur), es crearà un altre projecte. Per fer-ho, s'ha de clicar damunt del nom del projecte que apareix a la barra superior. S'ha de mostrar la següent pantalla:

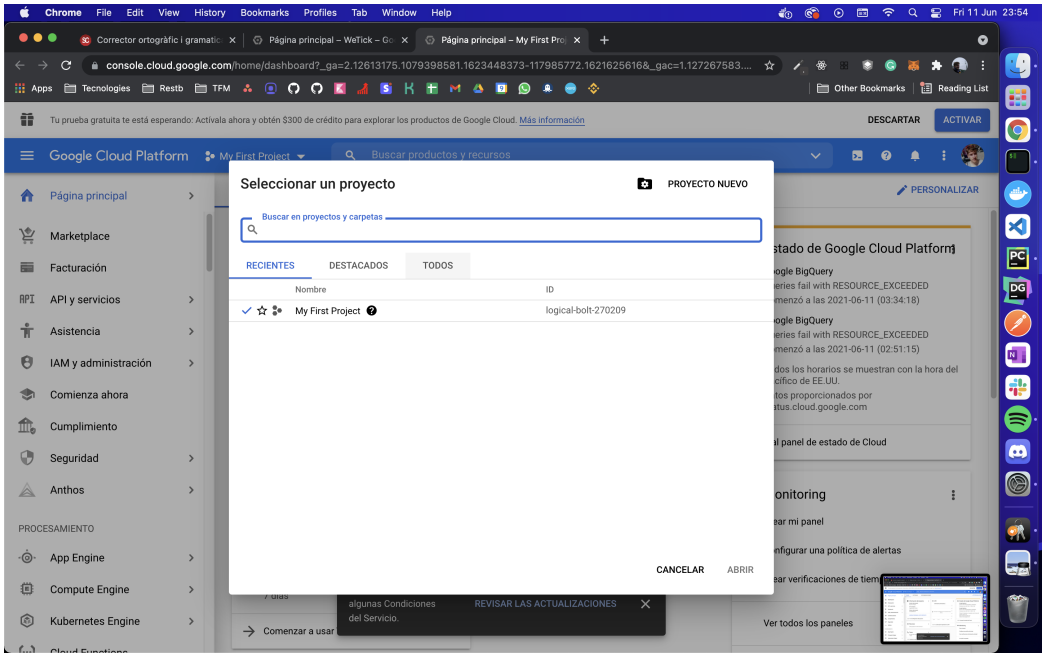


Fig. 6.21: Google Cloud: Projectes

En aquesta pàgina s'ha de donar clic a *Proyecto Nuevo*, a la part de dalt a la dreta del diàleg.

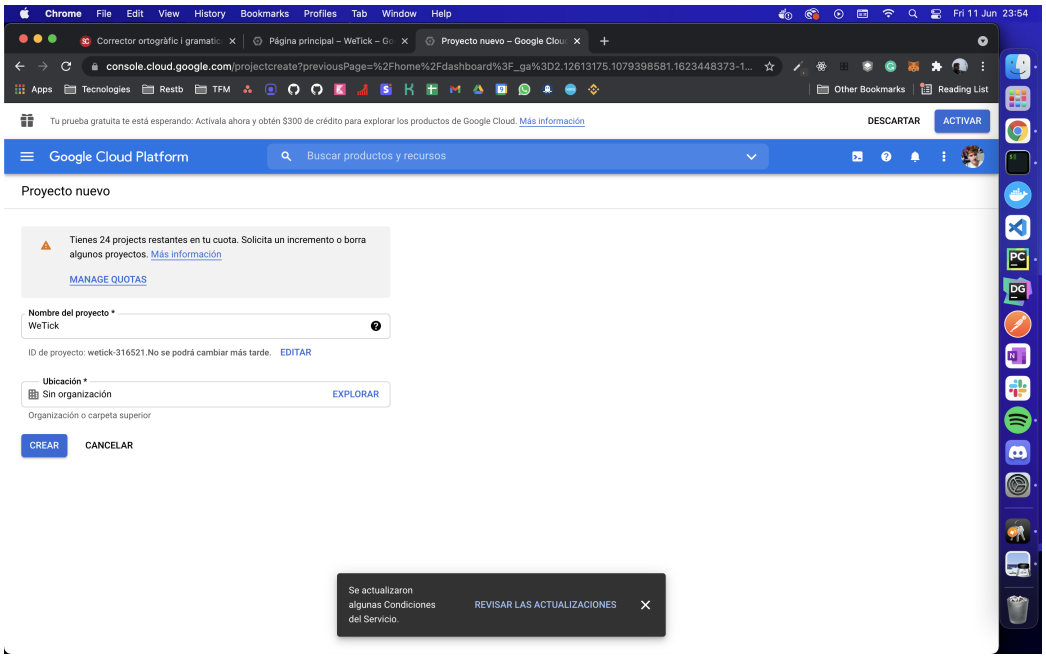


Fig. 6.22: Google Cloud: Nou Projecte

Quan es faci clic, es mostrarà una pàgina com l'anterior. En aquesta pàgina s'ha d'escriure el nom del nou projecte (en el meu cas he escrit *WeTick*), i a continuació, s'ha de clicar *Crear*. Al fer-ho es tornarà a mostrar la pàgina d'inici de *Google Cloud*, però amb la diferència, que ara es mostra tota la informació del projecte que s'acaba de crear.

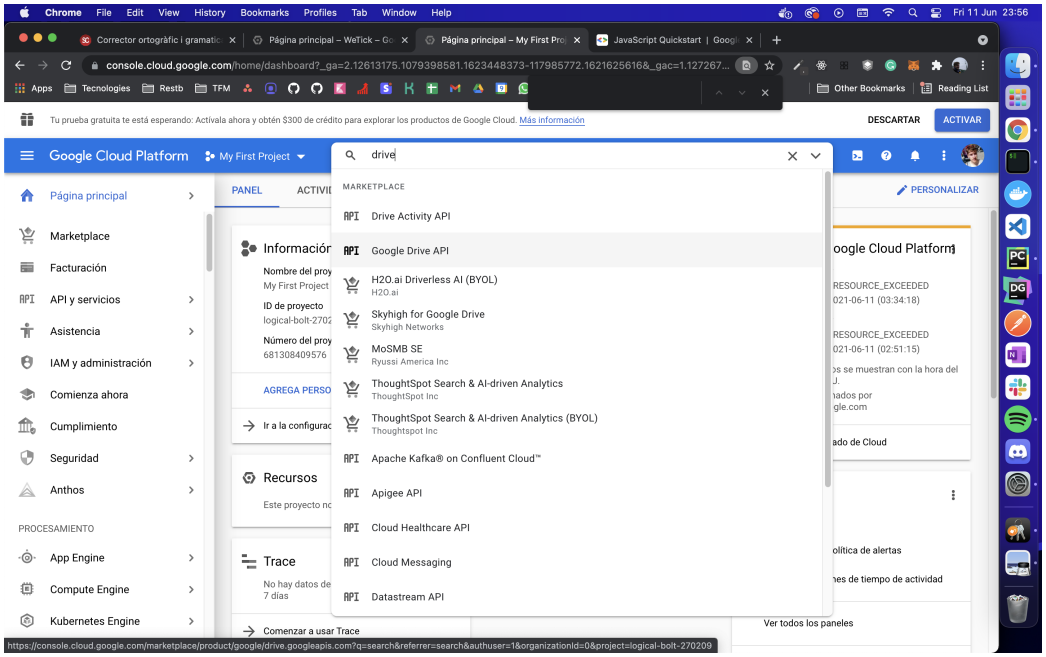


Fig. 6.23: Google Cloud: Pàgina d'inici 2

Des de la pàgina d'inici, s'ha d'escriure *drive* a la barra de cerca superior. Entre les opcions que apareixen s'ha de seleccionar *Google Drive APIs*. En fer-ho, es mostrarà la següent pantalla:

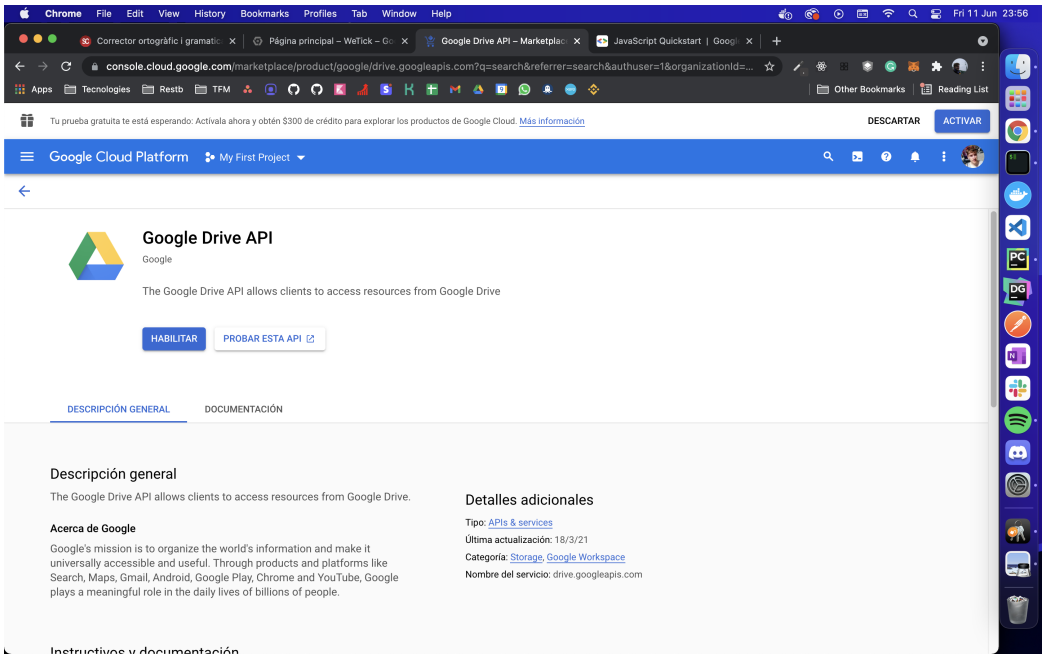


Fig. 6.24: Google Cloud: Google Drive API

En aquesta pàgina, s'ha de clicar *Habilitar*. Aquesta acció farà que l'API de *Google Drive* s'habiliti correctament i ja sigui accessible.

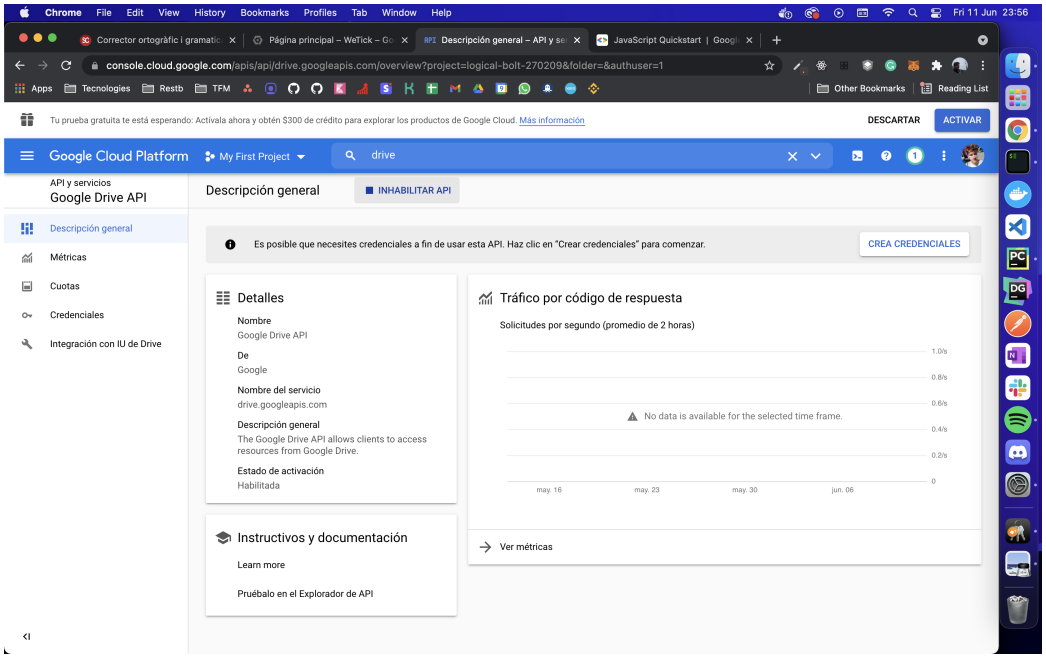


Fig. 6.25: Google Cloud: Google Drive API Pàgina d'inici

Quan s'habilita l'API de *Google Drive*, et porta a la pantalla d'inici d'aquesta. L'API està ja habilitada, però s'ha de configurar unes credencials per tal que aquesta sigui accessible. Per fer-ho, s'ha de clicar *Crear Credenciales*.

Configurar credencials OAuth

Al fer clic a *Crear Credenciales*, es mostrarà la següent pantalla:

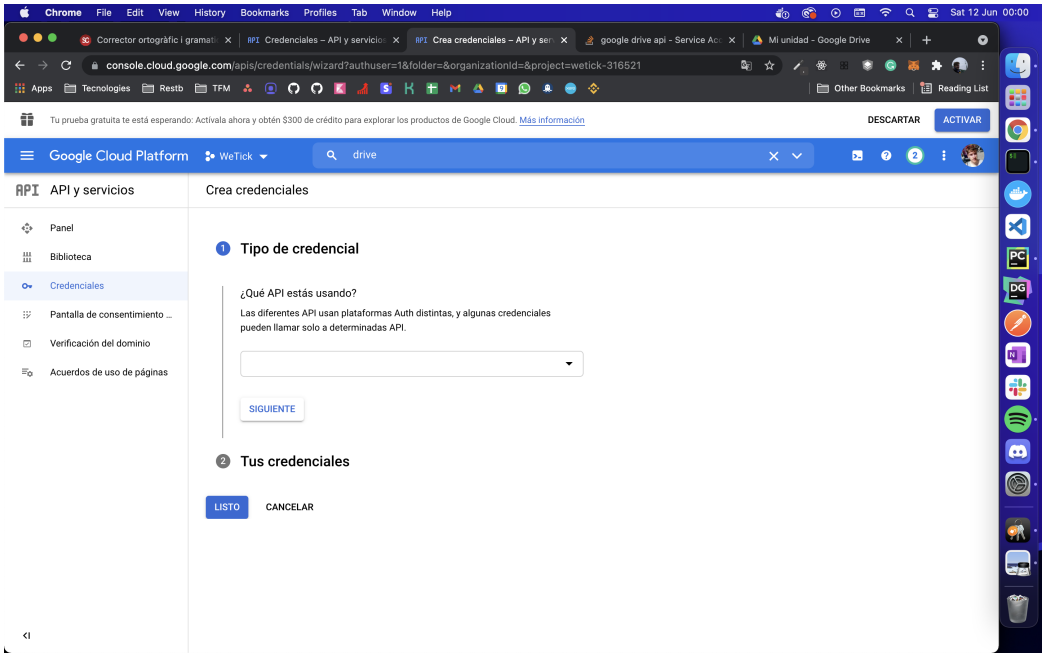


Fig. 6.26: Google Cloud: Configurar credencials

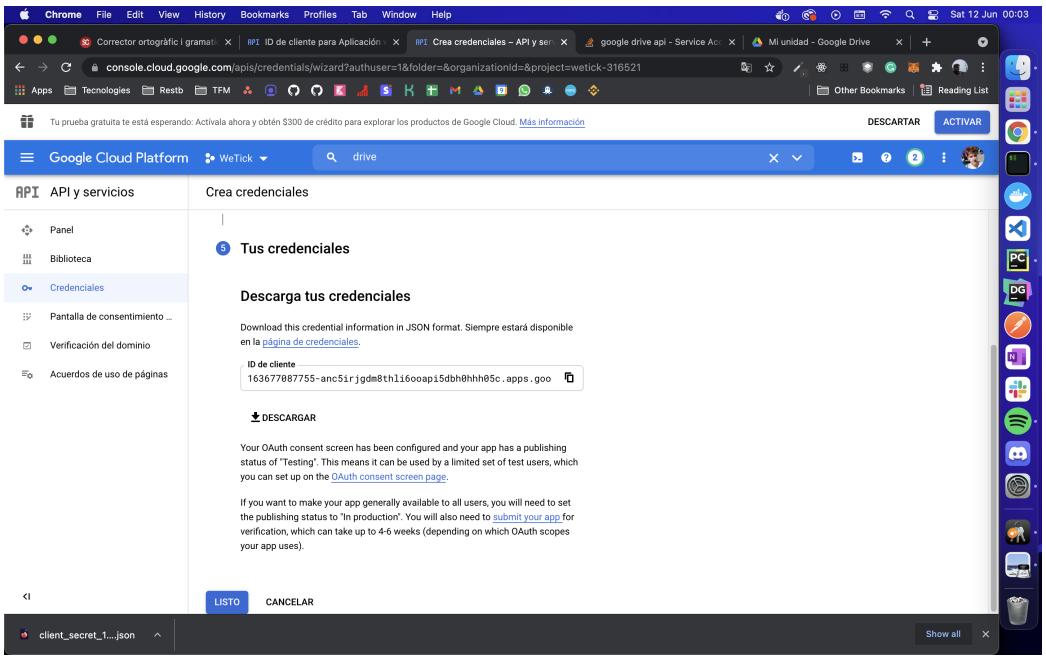


Fig. 6.27: Google Cloud: Configurar credencials 2

En aquesta pàgina, s'ha d'omplir tot el formulari amb tota la informació que es demana. A l'últim pas del formulari, apareixerà l'identificador de les credencials que es crearan, i un botó (*Descargar*) que descarrega un fitxar amb tota la informació de les credencials. S'ha de donar clic a *Descargar* i finalment, donar clic a *Listo*.

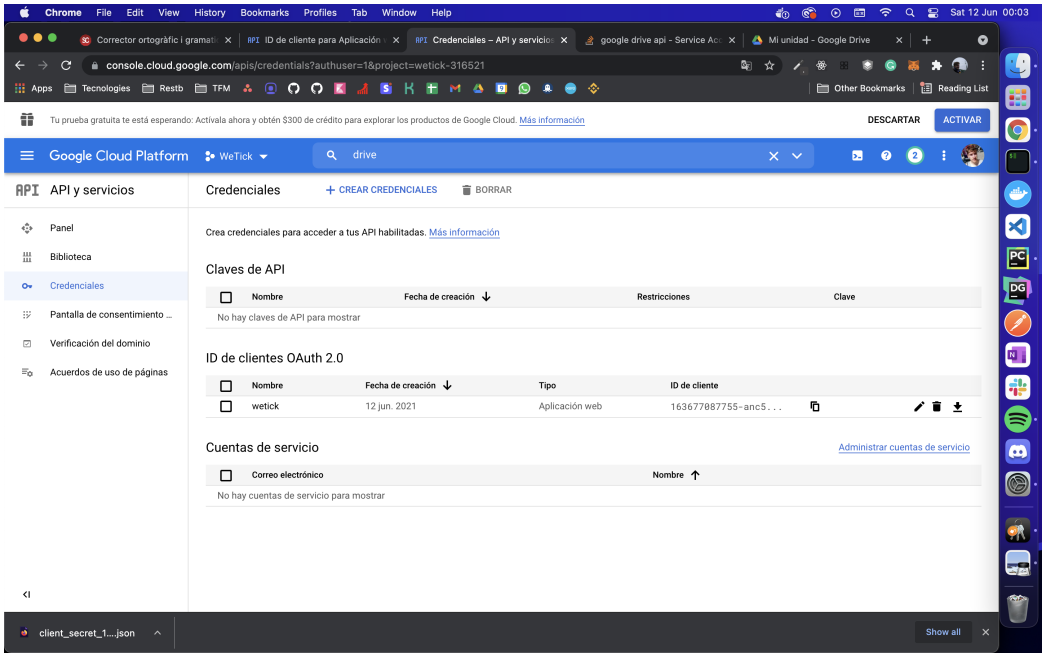


Fig. 6.28: Google Cloud: Pàgina de credencials

Quan s'ha acabat l'últim formulari, *Google Cloud* ens redirigeix a la pàgina de credencials. Una vegada arribats aquest punt, ja podem accedir a l'API a partir de la informació del fitxer que s'ha descarregat, i ja es pot implementar la pujada al servidor.

6.3.6 Pujada a Google Drive

Abans de començar a implementar aquest últim pas del servidor, s'ha de crear una carpeta a *Google Drive* la qual es farà servir per pujar tots els tiquets. Per identificar-la des del servidor, s'haurà de guardar el ser identificador, el qual el podem extreure de l'URL, i seria la cadena de caràcters que es troba per darrere de *folders/*.

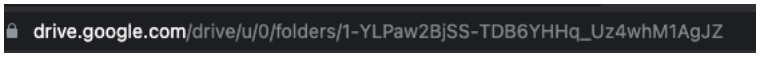


Fig. 6.29: Google Drive: URL directori de tiquets

En aquest cas, l'identificador del directori de *Google Drive*, seria *1-YLPaw2BjSS-TDB6YHHq_Uz4whM1AgJZ*.

Mòdul file_upload

Primerament, dins del mòdul *wetick*, es crearà un altre mòdul anomenat *file_uploader*. Dins d'aquest mòdul, es crearà un fitxer (uploader.py), el qual serà l'encarregat de pujar tots els tiquets firmats a *Google Drive*. Dins d'aquest fitxer, afegirem el següent contingut:

```
import random

from datetime import datetime
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive

def _get_file_title():
    time_value = datetime.now().timestamp()
    hash_value = random.getrandbits(128)
    file_name = f'{time_value}_{hash_value}.pdf'
    return file_name

def upload(upload_file):
    # OAUTH Drive
    g_auth = GoogleAuth('wetick/file_upload/settings.yaml')
    drive = GoogleDrive(g_auth)

    # Create File
    file_name = _get_file_title()
    g_file = drive.CreateFile({
        'parents': [{'id': '1-YLPaw2BjSS-TDB6YHHq_Uz4whM1AgJZ'}],
        'title': file_name
    })
    g_file.SetContentFile(upload_file)
    g_file.Upload()
    g_file.InsertPermission({
        'type': 'anyone',
        'value': 'anyone',
        'role': 'reader'
    })

    return g_file['webContentLink']
```

La funció principal d'aquest mòdul, és *upload_file()*. Aquesta funció fa els següents passos:

1. Autenticar-te a *Google Drive*.

2. Generar un nom aleatori basat en la data actual.

3. Pujar el document a *Google Drive*, a la carpeta que s'ha creat anteriorment.

4. Donar permisos de lectura a qualsevol persona que accedeixi a l'URL.

5. Retornar l'URL del tiquet.

En el mateix mòdul creem el fitxer *settings.yaml*, el qual conté la configuració per accedir a *Google Drive*. Tindrà el següent contingut:

```
client_config_backend: settings
client_config:
  client_id: ****
  client_secret: ****

save_credentials: True
save_credentials_backend: file
save_credentials_file: wetick/file_upload/credentials.txt

get_refresh_token: True
oauth_scope:
  - https://www.googleapis.com/auth/drive
  - https://www.googleapis.com/auth/drive.install
```

Per omplir els camps de *client_id* i *client_secret*, del fitxer, s'utilitzaran els valor especificats en el fitxer de credencials que s'ha descarregat anteriorment.

Finalment, s'ha de modificar el fitxer *listener.py*. Per una banda, s'ha d'importar el mòdul que s'acaba de crear:

```
from wetick.file_upload import uploader
```

I seguidament, modificar la funció *_on_create()*, la qual quedaria de la següent manera:

```
def _on_created(event):
```

```
socket_actions.manage_ticket()
file_path = signer.sign(event.src_path)
ticket_url = uploader.upload(file_path)
socket_actions.manage_ticket(ticket_url)
os.remove(file_path)
```

Una vegada s'han executat tots aquests passos, ja es pot executar el servidor. Quan es crei un nou fitxer al directori */var/spool/cups-pdf/ANONYMOUS*, el mòdul de *file_listener* el detectarà i enviarà un missatge pel *socket* informant que s'ha començat a gestionar un nou tiquet. Seguidament, aquest l'enviarà al mòdul de *file_signer* per a certificar el document. A continuació, s'enviarà al mòdul de *file_uploader* per pujar-lo a *Google Drive* i agafar l'URL. Finalment, s'envia un altre missatge pel *socket* amb l'URL del tiquet i aquest s'esborra del servidor.

La primera vegada que es gestiona un tiquet, s'obrirà una pestanya al navegador per tal que s'autoritzi la pujada de documents a *Google Drive*. S'ha d'acceptar tot. Una vegada es faci, es generarà el fitxer *credentials.txt* dins del mòdul *file_upload*.

Finalement, el servidor, quedaria amb la següent estructura:

```
api/
├── requeriments.txt
├── requeriments.lock
├── .gitignore
└── wetick
    ├── enums
    │   └── status_enum.py
    ├── file_listener
    │   └── listener.py
    ├── file_sign
    │   ├── signer.py
    │   └── wetick.p12
    ├── file_upload
    │   ├── credentils.txt
    │   ├── settings.yaml
    │   └── uploader.py
    ├── socket
    │   └── socket_actions.py
    ├── __main__.py
    └── __wetick__.py
```

6.4 Implementació de l'aplicació web

Ara que el servidor ja funciona i gestiona tots els tiquets correctament, es crearà una petita aplicació web amb *Angular*, que permetrà que els usuaris puguin descarregar el tiquet en el seu dispositiu mòbil a partir de l'URL.

6.4.1 Node i Npm

Abans de començar a programar l'aplicació web amb *Angular*, s'ha d'instal·lar *Node* i *npm*. Per fer-ho, s'ha d'anar a la pàgina de **Node** i descarregar la versió per al teu sistema operatiu.

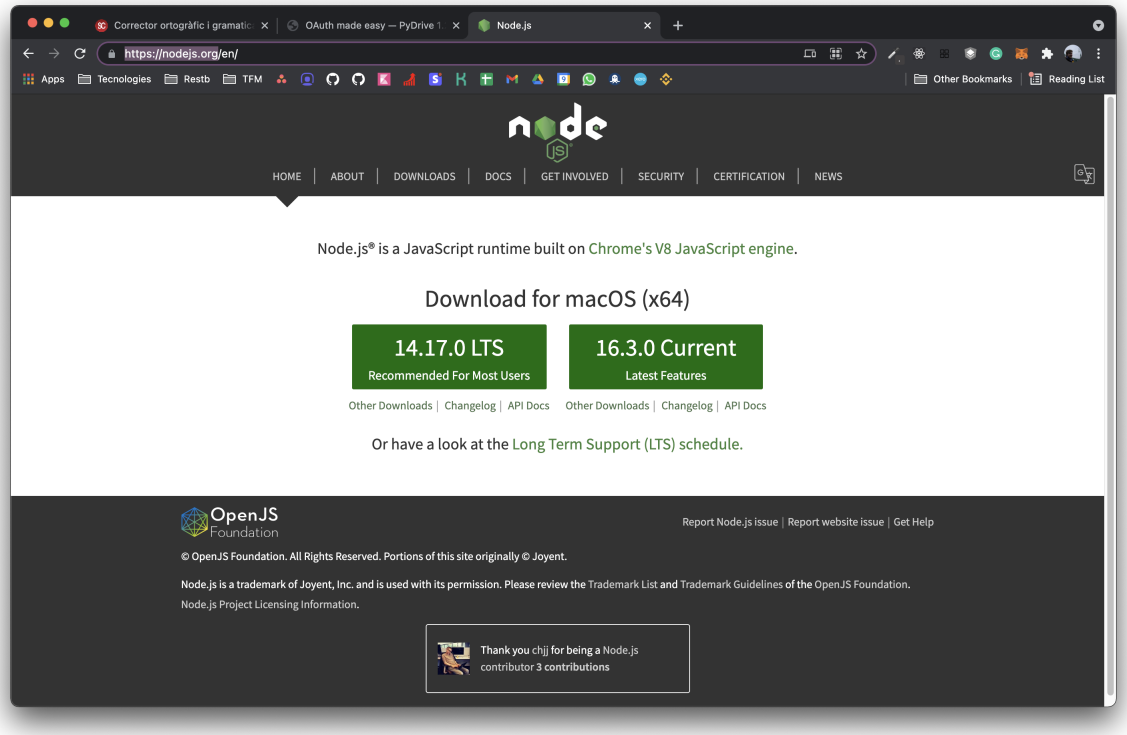


Fig. 6.30: Pantalla d'inici de Node.js

Instal·lació Angular

Quan *Node* i *npm* estan instal·lats, s'ha d'executar la següent comanda per tal d'instal·lar *Angular*:

```
$ npm install -g @angular/cli
```

6.4.2 Angular

Ara que *Angular* ja s'ha instal·lat al PC, és hora de crear l'aplicació web.

Creació del projecte

Per fer-ho, ens hem de dirigir a la carpeta on es troba el repositori de *git*, i seguidament a la subcarpeta *web*. Des d'aquesta carpeta, s'obre una terminal i s'exceuta la següent comanda:

```
$ ng new wetick --directory=./
```

Aquesta comanda, crearà un projecta d'*Angular* en el directori que ens trobém. Durant la seva execusió, apareixeràn vàries preguntes.

```
$ Would you like to add Angular routing? (y/N)
```

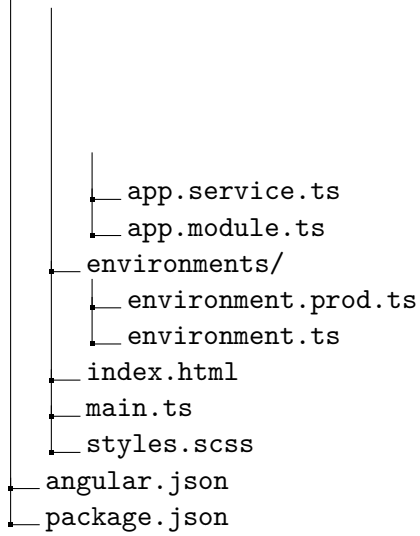
A la primera pregunta s'ha de respondre que si, ja que mai es sap si l'aplicació acabarà creixent.

```
$ Which stylesheet format would you like to use?
  CSS
  SCSS   [ https://sass-lang.com/documentation/syntax#scss                ]
  Sass   [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less   [ http://lesscss.org                                             ]
  Stylus [ http://stylus-lang.com                                         ]
```

La segona pregunta, s'ha de respondre la segona opció (*SCSS*), la qual afegeix la sintaxi *SCSS* al projecte en comptes de *CSS*.

Seguidament, el projecte d'*Angular* s'acabarà de crear. Dins del directori web, quedarà la següent estructura (només es mostren els elements a ressaltar):

```
web/
└── src/
    └── app/
        ├── app.component.html
        ├── app.component.scss
        ├── app.component.ts
        └── app.routing-module.ts
```



Aquests fitxers realitzen les següents funcions:

1. **package.json**: Conté els *scripts* per executar el projecte i totes les dependències d'aquest.

2. **angular.json**: Conté un *JSON*, que especifica totes les accions que s'han de dur a terme cada vegada que s'executa el projecte o es fa una *build*.

3. **styles.scss**: Conté els estils generals del projecte (Inicialment està buit).

4. **main.ts**: Fitxer *typescript* d'inici del projecte.

5. **index.html**: Conté informació general del projecte, com ara el títol i la icona que es mostren al navegador.

6. **app.module.ts**: Conté tots els mòduls d'*Angular* que s'utilitzen en el projecte.

7. **app.component**: Conté el component que es mostra per pantalla, amb les seves definicions de *html*, *scss* i *typescript*.

8. **app.service.ts**: Conté el servei que s'utilitzarà en un futur per comunicar l'aplicació amb el servidor.

9. **app.routing-module.ts**: Conté totes les rutes que es poden accedir des de l'aplicació web. En el meu cas està buit, ja que només em centraré en la ruta principal.

Afegir Angular Material

Per afegir *Angular Material* al projecte, s'ha d'executar la següent comanda des de la linia de comandes (al directori *web/*):

```
$ ng add @angular/material
```

De la mateixa manera que en la comanda per crear el projecte d'*angular*, farà unes quantes preguntes per configurar la llibreria.

```
$ Choose a prebuilt theme name, or "custom" for a custom theme: (Use arrow keys)
  Indigo/Pink        [ Preview: https://material.angular.io?theme=indigo-pink ]
  Deep Purple/Amber  [ Preview: https://material.angular.io?theme=deeppurple-amber ]
  Pink/Blue Grey     [ Preview: https://material.angular.io?theme=pink-bluegrey ]
  Purple/Green       [ Preview: https://material.angular.io?theme=purple-green ]
  Custom
```

La primera, pregunta quin estil es vol utilitzar per a l'aplicació. En el meu cas, he triat *Pink/Blue Grey*, per es pot elegir qualsevol opció.

```
$ Set up global Angular Material typography styles? (y/N)
```

La segona pregunta, demana si es vol afegir tipografies globals basades en la llibreria. En aquest cas he respost que sí.

```
$ Set up browser animations for Angular Material? (Y/n)
```

Finalment, es pregunta si volem afegir *browser animations*. També li he dit que si.

Aquest apartat és completament opcional, però per tal que l'aplicació web tingui una mica de cara i ulls de forma fàcil, és una bona elecció.

Instalar @techiediaries/ngx-qrcode

Per instal·lar la llibreria, des del directori *web/*, s'executarà la següent comanda per la línia de comandes:

```
$ npm install @techiediaries/ngx-qrcode --save
```

Una vegada executada, la llibreria ja està instal·lada. Tenir en compte que el paràmetre *–save* serveix perquè la dependència de la llibreria es guardi automàticament al fitxer *package.json*.

Implementació del component

Per implementar el component, s'obrirà el projecte del directori *web/* amb *Visual Studio Code*. Una vegada obert, s'obrirà el fitxer *app.module.ts* i s'importarà les següents dependències:

```
import { MatProgressSpinnerModule } from '@angular/material/progress-spinner';
import { MatToolbarModule } from '@angular/material/toolbar';
import { MatCardModule } from '@angular/material/card';
import { NgxQRCodeModule } from '@techiediaries/ngx-qrcode';
```

I seguidament, s'afegiran els mòduls importats a l'apartat *imports*:

```
MatProgressSpinnerModule,
MatToolbarModule,
MatCardModule,
NgxQRCodeModule
```

Seguidament, s'obrirà el fitxer *app.component.html* i s'afegirà el següent contingut:

```
<mat-toolbar color="primary" class="toolbar">
  <span>Wetick</span>
</mat-toolbar>
<div *ngIf="ticketStatus === 'NoTicket'" class="full-width-div">
  <mat-card class="ticket-card">
    <h2>There is not tickets to show... </h2>
  </mat-card>
</div>
<div *ngIf="ticketStatus === 'TicketProcessing'" class="full-width-div">
  <mat-card class="ticket-card">
    <h2>Your ticket is been processed... Please wait </h2>
    <mat-spinner class="ticket-processing-spinner"></mat-spinner>
  </mat-card>
</div>
<div *ngIf="ticketStatus === 'TicketProcessed'" class="full-width-div">
  <mat-card class="ticket-card">
    <h2>Your ticket has been processed successfully! </h2>
    <h4>Use the following QR Code to download it:</h4>
    <ngx-qrcode
```

```
      [elementType]="elementType"
      [errorCorrectionLevel]="correctionLevel"
      [value]="url"
      [width]="250"></ngx-qrcode>
  </mat-card>
</div>
```

A continuació s'obre el fitxer *app.component.scss* i s'afegeix el següent contingut:

```
.toolbar {
  position: absolute;
  z-index: 1;
}

.full-width-div {
  width: 100%;
  height: 100vh;
  display: flex;
  justify-content: space-around;

  .ticket-card {
    min-width: 25%;
    min-height: 50%;
    margin: auto;
    display: flex;
    flex-direction: column;
    align-items: center;

    .ticket-processing-spinner {
      margin-top: 5rem;
      margin-bottom: 2rem;
      width: 250px;
      height: 250px;
    }
  }
}
```

Finalment, s'obre el fitxer *app.component.ts* i s'afegeixen les següents dependències:

```
import { Component } from '@angular/core';
```

```
import {
    NgxQrcodeElementTypes,
    NgxQrcodeErrorCorrectionLevels
} from '@techiediaries/ngx-qrcode';
```

I el següent contingut dins de la classe del component:

```
// QR Management
elementType = NgxQrcodeElementTypes.URL;
correctionLevel = NgxQrcodeErrorCorrectionLevels.HIGH
url = null

constructor() {}
```

Arribats a aquest punt, es té tota la lògica implementada. Per tant, si s'executa l'aplicació, es veurà una pantalla com aquesta en el port 4200 (que és el port per defecte d'*Angular*).

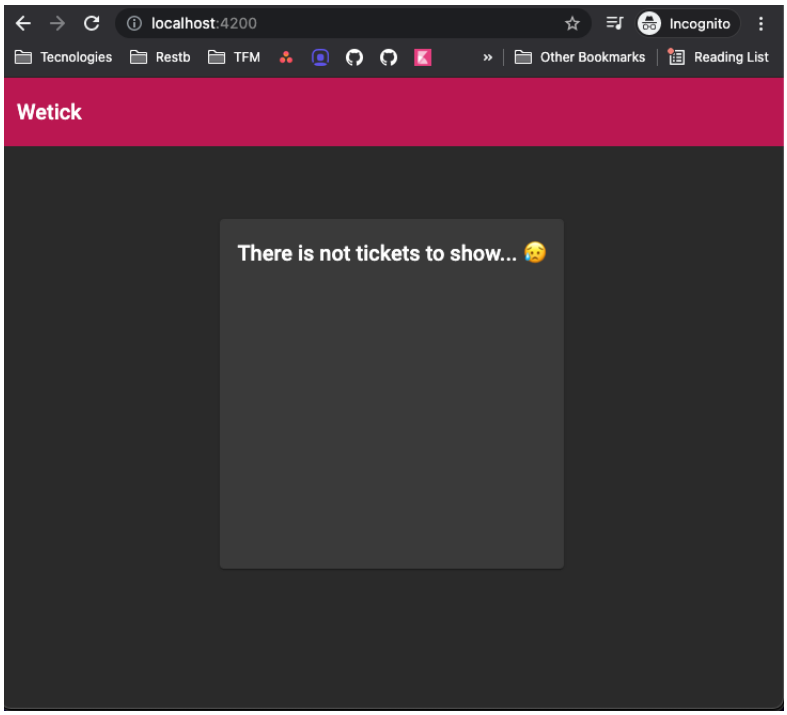


Fig. 6.31: Aplicació web 1

Per executar el projecte d'*Angular*, s'ha d'executar la següent comanda dins del directori *web/*:

```
$ npm start
```

6.4.3 SocketIO

Per acabar l'aplicació d'*Angular*, s'ha d'implementar la comunicació amb el *socket* del servidor de *Flask*.

Instal·lació SocketIO

Per instalar *SocketIO* al projecte, s'ha d'executar la següent comanda des de la linia de comandes (al directori *web/*):

```
$ npm install --save ngx-socket-io
```

Implementació del servei

Per començar, es modificaran els dos fitxers que es troben dins del directori *environment/* i se'ls hi ha d'afegir l'URL a la qual funciona el *socket*. El contingut quedarà així:

```
export const environment = {
  url: 'http://localhost:5000'
};
```

Seguidament, s'ha d'obrir el fitxer *app.modules.ts* i afegir els següents imports i la següent constant:

```
import { SocketIoConfig, SocketIoModule } from 'ngx-socket-io';
import { environment } from '../environments/environment';

const config: SocketIoConfig = { url: environment.url, options: {} };
```

I els mòduls pertinents a *SocketIO* dins de la secció *imports*:

```
SocketIoModule.forRoot(config),
```

Seguidament, s'ha d'obrir el fitxer *app.service.ts*, i modificar el contingut de forma que quedi de la següent manera:

```
import { Injectable } from '@angular/core';
import { Socket } from 'ngx-socket-io';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class WetickService {

  constructor(private socket: Socket) { }

  getMessage() {
    return new Observable((observer) => {
      this.socket.on('message', (message) => {
        observer.next(message)
      })
    })
  }
}
```

Aquest servei és el que es comunica directament amb el *socket* implementat en el servidor, a partir de la funció *getMessage()*. Aquesta funció escolta tot el temps a l'espera de missatges que puguin arribar pel *socket*. Tot i això, com que la funció no s'utilitza en cap lloc, cada vegada que arriba un missatge, no fa cap acció. Per tal que mostri l'URL o l'estat de la gestió del tiquet, es modificarà el fitxer *app.component.ts* i s'afegirà la crida a aquesta funció.

En primer lloc, s'ha d'importar el servei que s'acaba d'implementar:

```
import { WetickService } from './app.service';
```

Seguidament, s'ha de modificar la funció *constructor()* per tal que quedi de la següent manera:

```
constructor(wetickService: WetickService) {
    wetickService.getMessage().subscribe(message => {
      this.ticketStatus = message['status'];
      if (this.ticketStatus === 'TicketProcessed') {
        this.url = message['url'];
      } else {
        this.url = null;
```

```
      }
    });
  }
```

En aquest fragment de codi, fem que el component es subscrigui a la funció *getMessage()* del servei, per tal que cada vegada que hi hagui un missatge nou al *socket*, la funció canviarà l'estat del component mostrant el codi QR que conté l'URL, o fent-li saber a l'usuari que un tiquet s'està gestionant.

Si s'executa el servidor i l'aplicació web a la vegada, cada vegada que es gestioni un tiquet, es mostrarà la següent pantalla:

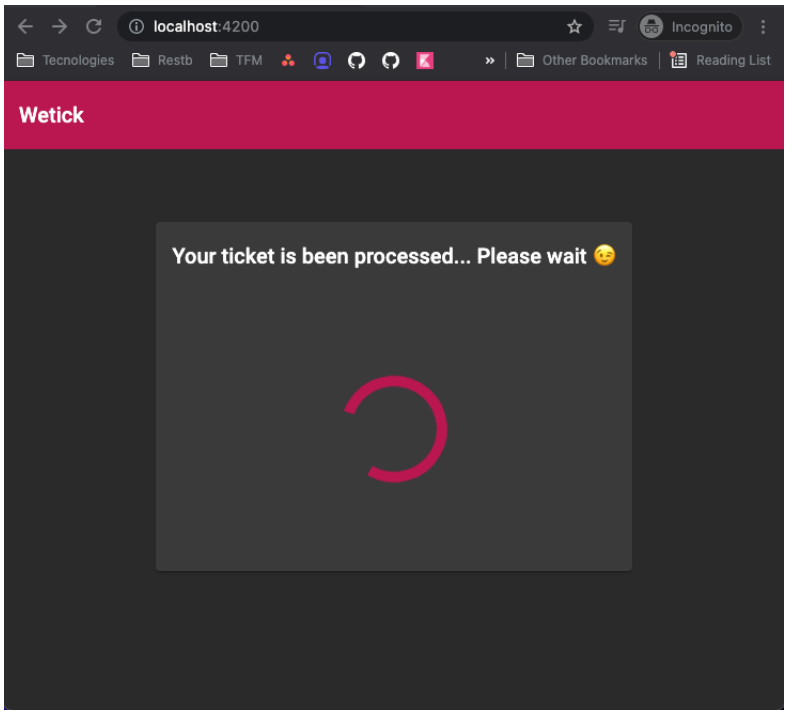


Fig. 6.32: Aplicació web 2

I quan el tiquet s'hagi gestionat i pujat a *Google Drive* es mostrarà la següent pantalla, la qual mostra un codi QR que descarrega el tiquet una vegada s'escaneja:

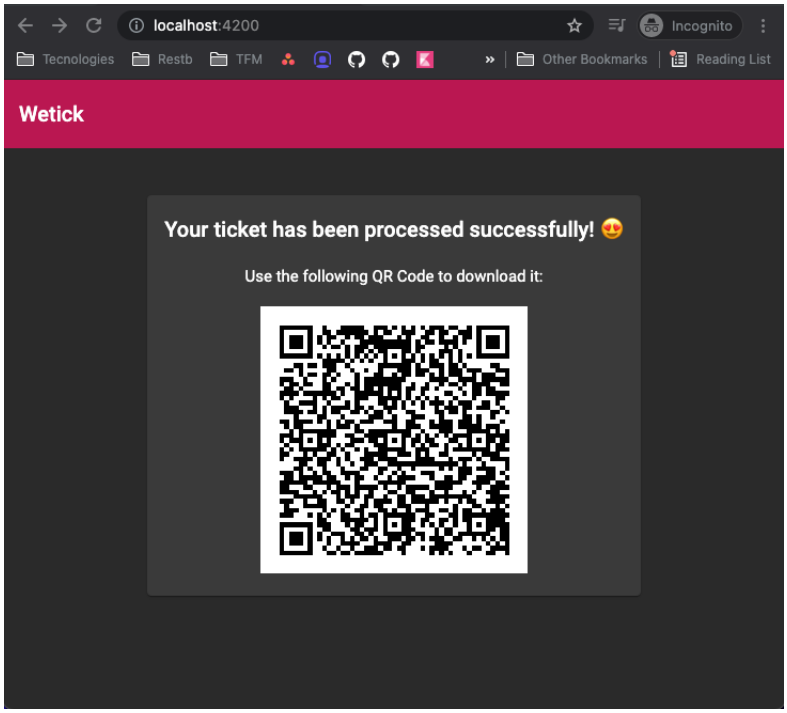


Fig. 6.33: Aplicació web 3

Com que executar els dos programes a la vegada és bastant farragós, i no s'inicien automàticament quan s'encén el PC o la *RaspberryPi*, s'ha de crear un fitxer *Docker* per configurar-ho.

6.5 Docker

Docker és un programa que automatitza el desplegament d'aplicacions en el nostre PC. En aquest cas, s'utilitzarà per automatitzar l'execució del servidor i l'aplicació web en la *RaspberryPi*.

6.5.1 Instal·lació Docker

Per instal·lar *Docker*, s'ha d'anar a la **pàgina de Docker** i seguidament descarregar el *software* de *Docker Desktop*. Seguidament s'ha d'instal·lar al PC i a la *RaspberryPi*.

6.5.2 Dockerització del projecte

Per tal d'executar el projecte dins d'un *Docker*, s'han de crear tres fitxers dins del directori principal del repositori.

Dockerització del servidor

Per *dockertizar* el servidor, s'ha d'anar a la subcarpeta *api/* i crear el fitxer *uwsgi.ini*. El contingut que ha de tenir és el següent:

```
[uwsgi]
module = wetick.wetick
callable = app
enable-threads = true
threads = 4
lazy-apps = true
die-on-term = true
buffer-size = 32768
```

Seguidament es crea el fitxer *Dockerfile* amb el següent contingut:

```
# Import base image
FROM tiangolo/uwsgi-nginx-flask:python3.7
ENV HOME /srv/wetick

# Set up configuration
ENV UWSGI_CHEAPER 1
ENV UWSGI_PROCESSES 2
ENV NGINX_MAX_UPLOAD 10m
ENV UWSGI_INI ${HOME}/uwsgi.ini

# Add needed files
ADD ./wetick ${HOME}/wetick
ADD ./requirements.lock ${HOME}/requirements.lock
ADD ./uwsgi.ini ${HOME}/uwsgi.ini

# Install dependencies
RUN pip install --upgrade pip
RUN pip install -r ${HOME}/requirements.lock

# Set up port to listen
ENV LISTEN_PORT 5000
EXPOSE 5000
WORKDIR ${HOME}
```

En el fitxer *uwsgi.ini*, es configura com volem que s'executi el servidor, mentre que al fitxer *Dockerfile* es configura totes les comandes que s'han d'executar per tal de compilar el servidor correctament.

Dockerització de l'aplicació web

Per *dockertizar* l'aplicació web, s'ha d'anar a la subcarpeta *web/* i crear el fitxer *Dockerfile*. El contingut que ha de tenir és el següent:

```
FROM node:13.12.0-alpine
RUN mkdir -p /app
WORKDIR /app
COPY package.json /app
RUN npm install -g serve
RUN npm install
COPY . /app
RUN npm run build --prod
EXPOSE 4200
CMD ["serve", "-s", "-l", "4200", "./dist/wetick"]
```

En aquest fitxer, s'especifica quina versió de node es vol instalar, juntament amb totes els passos que s'han de serguir per aixecar l'aplicació web.

Docker Compose

Finalment, al directori base del repositori, s'ha de creat el fitxer *docker-compose.yml*, i se li afegirà el següent contingut:

```
version: '3.5'
services:

  web:
    image: wetick_web
    container_name: wetick_web
    build: web
    networks:
        - wetick
    ports:
        - 4200:4200
```

```
    environment:
        - DEVELOPMENT_MODE=true
    restart: always
    depends_on:
      - api

  api:
    image: wetick_api
    container_name: wetick_api
    build: api
    networks:
        - wetick
    ports:
        - 5000:5000
    environment:
        - DEVELOPMENT_MODE=true
    restart: always

networks:
  wetick:
    name: wetick
    driver: bridge
```

Aquest fitxer conté l'ordre en què es volen instal·lar les diferents parts del projecte. El servei *web*, depèn del servei *api*. També s'especifica en quin directori es troba el servei amb la clau *build*. Finalment, també es diu quins ports es volen tenir oberts per tal d'utilitzar els serveis.

Mirgació a la RaspberryPi

Finalment, ara que tot el projecte ja està desenvolupat, toca migrar tot el codi dins de la *RaspberryPi*. Per fer-ho, es seguiràn els següents passos:

1. Instal·lar el *Docker Desktop* a la *RaspberryPi*.

2. Instal·lar *git* a la RaspberryPi.

3. Assegurar-se que tot el codi que s'ha desenvolupat al PC estigui dins del repositori de *git*.

4. Des de la *RaspberryPi*, descarregar el repositori de *GitHub*.

5. Entrar al directori base del repositori, i executar la següent comanda amb la línia de comandes:

```
$ docker-compose -f docker-compose.yml up -d --build web
```

Fet això, la RaspberryPi ja conté tot el software funcionant. A més, cada vegada que es reinicïi el software es tornarà a aixecar automàticament sense la necessitat de fer res.

7 Funcionament

El funcionament del sistema, tracta en substituir la impressora de tiquets dels caixers per una impressora virtual. Per fer-ho, tal com s'explica en l'apartat anterior, s'utilitza una *RaspberryPi 3* i *Google Drive*.

Els usuaris que faran ús d'aquest sistema serien els següents:

- **Venedor**: És l'encarregat de vendre els productes d'una empresa i el que crearà el tiquet dels productes que es compren.

- **Client**: És algú que compra o lloga un servei. Aquest, per normativa ha de tenir dret a rebre un tiquet com a comprovant de la compra.

Per tal que els usuaris del sistema, continuïn satisfent les necessitats esmentades, el funcionament del sistema, seguirà el següent esquema:

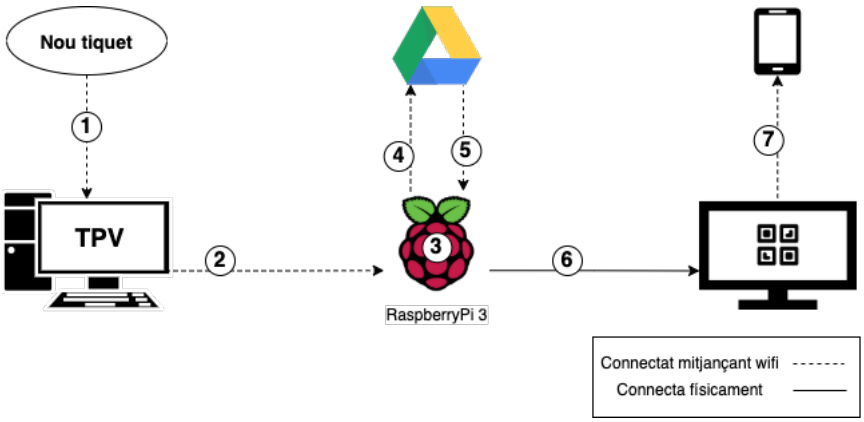


Fig. 7.1: Diagrama del funcionament

1. El venedor crea un tiquet mitjançant el TCP, on es reflecteixen tots els productes o serveis que el client ha comprat.

2. Quan el TCP imprimeix el tiquet, aquest s'envia en un directori concret de la *RaspberryPi 3*, en format PDF.

3. Quan la *RaspberryPi* detecta el document, el signa i li canvia el nom per un que no sigui deduïble.

4. Seguidament la *RaspberryPi,* puja el document signat a *Google Drive* i li dóna permisos de lectura a tothom que hi vulgui accedir.

5. A continuació la *RaspberryPi*, recupera l'URL on s'ha pujat el document.

6. Aquest URL, es mostra per la pantalla que està connectada a la *RaspberryPi*, en format de codi QR.

7. Finalement, quan el client escaneja el codi QR amb el seu dispositiu mòbil, el tiquet es descarregarà.

8 Planificació Temporal

8.1 Fase d'Anàlisis

La fase d'anàlisis ha tingut una durada d'un mes (16 de febrer - 15 de març). Aquesta fase ha consistit en fer una anàlisi de la utilització dels tiquets en el nostre dia a dia i l'impacte que causen en el medi ambient. A més, també s'han buscat possibles solucions al problema.

8.2 Fase d'Implementació

La fase d'implementació ha tingut una durada de dos mesos i mig (16 de març - 24 de maig). Durant aquesta fase d'implementació, s'han dut a terme dos projectes per separat. El primer, el qual consistia en crear un TPV (Terminal Punt de Venda) i té una durada d'un mes i mig, no es va acabar, ja que es va considerar amb el tutor que no es tenia el coneixement necessari per dur a terme el projecte. El segon, el qual consisteix en utilitzar una RaspberryPi com a impressora per digitalitzar els tiquets, és el que s'ha acabat presentant com a projecte, i ha tingut una durada d'un mes.

8.2.1 Implementació de TPV

La durada d'aquesta part de la fase d'implementació ha sigut d'un mes i mig (16 de març - 26 d'abril). Aquesta part s'ha dividit en 3 sprints de dues setmanes cada un, i després es va descartar per les raons esmentades anteriorment.

Sprint 1

Durant el primer sprint (16 de març - 29 de març), es va planificar l'arquitectura del sistema. Vaig decidir que l'aplicació tindria les següents parts: Base de dades (on es guardaria tota la informació), aplicació backend (la qual ens serviria per gestionar tota la base de dades i servir tota la informació necessària), aplicació web (la qual es faria servir com a TPV i es comunicaria amb l'aplicació de backend), i finalment, una aplicació mòbil (on els usuaris podrien veure tots els tiquets gestionats des de l'aplicació web, i que també es comunica amb el backend).

Sprint 2

El segon sprint (30 de març - 12 d'abril), va consistir en crear la base de dades, i implementar part de l'aplicació de backend, necessària per poder gestionar tots els tiquets des de l'aplicació web i visualitzar-los en l'aplicació mòbil en un futur.

Sprint 3

El tercer sprint (13 d'abril - 26 d'abril), va consistir en començar a desenvolupar l'aplicació web per gestionar els tiquets. Arribats en aquest punt, es va descartar tot el treball fet i es va començar el projecte d'implementar una RaspberryPi com a impressora.

8.2.2 Implementació d'una RaspberryPi com a impressora

La durada d'aquesta part de la fase d'implementació ha sigut d'un mes (27 d'abril - 24 de maig). Aquesta part s'ha dividit en 4 sprints d'una setmana, i s'ha procedit de la següent manera:

Sprint 1

El primer sprint (27 d'abril - 3 de maig), va consistir planificar els passos que s'havien de fer per dur a terme el projecte, i en configurar la RaspberryPi per tal que es detectés com una impressora dins de la xarxa wifi.

Sprint 2

El segon sprint (4 de maig - 10 de maig), va consistir en capturar tots els documents que s'enviaven per imprimir a la RaspberryPi i enviar un missatge aleatori per un socket cada vegada que es detectava el document.

Sprint 3

El tercer sprint (11 de maig - 17 de maig), va consistir en pujar al núvol els documents detectats en el pas anterior. Una vegada els documents s'han pujat, enviar l'URL del document pel socket que s'ha esmentat en l'sprint 2. Addicionalment, també vaig implementar una aplicació web que agafa l'URL del socket, i la mostra en format QR per una pantalla.

Sprint 4

El primer sprint (18 de maig - 24 de maig), va consistir en firmar tots els documents digitalment abans de pujar-los al núvol, i configurar tota l'aplicació en un docker (per tal que cada vegada que s'engegui la RaspberriPi, l'aplicació s'executi automàticament).

8.3 Fase de Redacció

La fase de redacció ha tingut una durada de quatre setmanes (25 de maig - 21 de juny). Aquesta fase ha consistit a redactar tota la memòria del sistema que s'ha desenvolupat, i els resultats de l'anàlisi inicial.

8.4 Fase de Tancament

La fase de tancament ha tingut una durada d'una setmana (22 de Juny - 28 de juny). Aquesta fase ha consistit en preparar la defensa del projecte.

8.5 Diagrama de Gantt

A continuació, es mostra el diagrama de Gantt del projecte.

Tasca	Data d'inici	Data de finalització	16/02/2021 - 22/02/2021	23/02/2021 - 01/03/2021	02/03/2021 - 08/03/2021	09/03/2021 - 15/03/2021	16/03/2021 - 22/03/2021	23/03/2021 - 29/03/2021	30/03/2021 - 05/04/2021	06/04/2021 - 12/04/2021	13/04/2021 - 19/04/2021	20/04/2021 - 26/04/2021	27/04/2021 - 03/05/2021	04/05/2021 - 10/05/2021	11/05/2021 - 17/05/2021	18/05/2021 - 24/05/2021	25/05/2021 - 31/05/2021	01/06/2021 - 07/06/2021	08/06/2021 - 14/06/2021	15/06/2021 - 21/06/2021	22/02/2021 - 28/06/2021
Fase d'Ànalisis	16/02/2021	15/03/2021																			
Fase d'Implementació	16/03/2021	24/05/2021																			
Implementació de TPV	16/03/2021	26/04/2021																			
Sprint 1	16/03/2021	29/03/2021																			
Sprint 2	30/03/2021	12/04/2021																			
Sprint 3	13/04/2021	26/04/2021																			
RaspberryPi com a impressora	27/04/2021	24/05/2021																			
Sprint 1	27/04/2021	03/05/2021																			
Sprint 2	04/05/2021	10/05/2021																			
Sprint 3	11/05/2021	17/05/2021																			
Sprint 4	18/05/2021	24/05/2021																			
Fase de Redacció	25/05/2021	21/06/2021																			
Fase de Tancament	22/06/2021	28/06/2021																			

Fig. 8.1: Diagrama de Gantt

9 Pressupost

9.1 Consideracions Inicials

En aquesta secció, s'identifiquen tots els recursos necessaris per a la realització del projecte i es fa una estimació del cost total d'aquests, tenint en compte aquestes dues variants:

- S'assumeix que el projecte és universitari, i que per tant, no té inversió econòmica.
- S'assumeix que és un projecte competitiu, amb les hores de recursos humans a preu de mercat.

9.2 Identificació i Estimació de Costos

9.2.1 Costos Directes

Els costos directes per activitat en aquest projecte, inclouen els recursos humans involucrats en cada una de les activitats definides al diagrama de Gantt, presentat anteriorment. Encara que a la pràctica tot el projecte estigui desenvolupat per una sola persona, cada activitat d'aquest correspon a un rol diferent, i per tant, les hores tindran un preu de mercat diferent:

- Tasques de gestió i documentació (Cap de Projecte): 30€/h
- Implementació (Software Developer): 25€/h

A continuació, es mostra una taula amb el cost de mercat i el cost estimat d'aquest projecte:

Activitat	Unitats	Preu de mercat	Cost de mercat	Preu estimat	Cost Estimat
Fase d'Ànalisis	200h (4 setmanes)	30 €/h	6.000€	0 €/h	0€
Fase d'implementació	500h (10 setmanes)	25 €/h	12.500€	0 €/h	0€
Fase de Redacció	200h (4 setmanes)	30 €/h	6.000€	0 €/h	0€
Fase de Tancament	50h (1 setmana)	30 €/h	1.500€	0 €/h	0€
TOTAL	950h		26.000€		0€

Tab. 9.1: Costos Directes

9.2.2 Costos Indirctes

En aquesta secció es concideren tots els costos que afecten de forma indirecta, a l'hora de dur a terme el projecte. Són els següents:

- Ordinador portàtil: Suposant que la vida útil d'un portàtil és de 5 anys i que el 60% del seu ús durant els propers 4 mesos estarà dedicat al projecte.

- RaspberryPi 3: S'ha comprat una *RaspberryPi 3* per tal de dur a terme el projecte. El preu de la *RaspberryPi 3* és de 35€.[22]

- Connexió a internet: Tot i que la connexió a internet és imprescindible per quasi qualsevol informàtic, es considerarà que ja existeix una contractació. A partir d'aquí es suposarà que de l'ample de banda consumit, se'n dedica un 20% al projecte durant el temps que aquest duri.

Producte	Unitats	Preu unitari	Percentatge de dedicació	Cost estimat
Ordinador portàtil	4 mesos	600 €/mes	60%	24€
RaspberryPi 3	1	35€	---	35€
Connexió a internet	4 mesos	35,5 €/mes	15%	21,30€
TOTAL				80,30 €

Tab. 9.2: Costos Indirectes

9.2.3 Contingència

Per evitar qualsevol risc que hi pugui haver, es reservarà una part del pressupost per a la part de contingència. Aquest cost serà un 10% de la suma dels costos directes i indirectes.

Producte	Percentatge	Preu de mercat	Cost de mercat	Preu estimat	Cost Estimat
Costos directes	10%	26.000€	3.900€	0 €/h	0€
Costos indirectes	10%	80,30€	8,03 €	80,30€	8,03 €
TOTAL			3.908,03 €		8,03 €

Tab. 9.3: Contingència

9.2.4 Imprevistos

Durant el projecte es poden presentar diferents imprevistos que ens facin encarir el projecte, els quals són:

- Avaria ordinador: en cas que hi hagués algun problema amb el l'ordinador, s'hauria de reparar o comprar-ne un de nou, augmentant el cost del projecte en màxim el preu de l'ordinador nou. Assignarem a aquesta eventualitat una probabilitat del 5%.

Imprevist	Probabilitat	Unitats	Preu de mercat	Cost de mercat	Preu estimat	Cost Estimat
Avaria ordinador	5%	0h	40€/h	0€	0 €/h	0€
TOTAL				0€		0€

Tab. 9.4: Imprevistos

9.2.5 Còmput Final

Durant el projecte no es consideren les següents situacions:

- No es tenen en compte els augments de preu (escalació) durant el projecte, ja que es de curta durada i no es preveu que es doni la situació.

- No hi ha cap marge de benefici, ja que el projecte és completament sense ànim de lucre i no està destinat a la venda d'un producte.

Concepte	Cost de mercat	Cost Estimat
Costos directes	26.000€	0€
Costos indirectes	80,30 €	80,30 €
Contingència	3.908,03 €	8,03 €
Imprevistos	0€	0€
TOTAL	29.988€	88,33 €

Tab. 9.5: Còmput Final

10 Sostenibilitat i Compromís Social

10.1 Dimensió ambiental

L'impacte mediambiental del projecte, és negatiu, ja que s'estalvia la impressió dels tiquets en paper. Sí que és veritat, que per fabricar la RaspberryPi i transportar-la al lloc, té un cost energètic i un impacte mediambiental, però només una vegada.

Donat el sistema actual, es té una impressora per TPV (terminal punt de venda). Per tant, el cost energètic i l'impacte ambiental que s'ha de contar per cada TPV és el de fabricació, transport i la quantitat de residus generats per la impressora. Addicionalment, s'ha de contar el cost energètic i l'impacte ambiental per cada un dels tiquets que s'imprimeixen.

En el sistema que s'ha desenvolupat, també hi hauria un RaspberryPi 3 per TPV. Però a diferència del sistema actual, només hi ha el cost energètic i l'impacte ambiental de la RaspberryPi. Els tiquets que es guarden a Google Drive, també generen un cost, però aquest és negligible si el compares amb el cost que generen els tiquets en paper.

10.2 Dimensió social

En l'àmbit social, aquest projecte pot aportar un gran valor a les persones que estan acostumades a tractar amb la tecnologia, però possiblement, pot ser problemàtic per aquelles que no.

Tot i això, a causa de la crisi de la COVID-19, en la majoria de restaurants, s'ha instaurat la tecnologia dels codis QR, cosa que ha fet que moltes persones si haguessin d'adaptar ràpidament.

10.3 Dimensió econòmica

En l'àmbit econòmic, s'han de valorar dues coses: el cost per tiquet, i el cost del hardware.

Per la banda del hardware, una impressora térmica val entre uns 50€ i 500€, mentre que el cost de la RaspberryPi 3 (que s'ha utilitzat en aquest projecte), és d'uns 35€. Per tant, des de l'inici, ja hi ha un cost menor en instal·lar Raspberries en comptes d'impressores.

Per la banda del cost per tiquet, *Google Drive API*, té un límit de 1000000 de crides gratuïtes al dia. Ho sigui que es poden fer 1000000 de tiquets diaris de forma gratuïta. A més, els primers 15GB de tiquets són gratuïts, a partir d'aquí, s'ha de pagar una quota mensual de 1.99€. Només que es compari amb el preu que val un rotllo de paper de tiquet (2-5€), ja hi ha un cost menor.[23]

11 Conclusions

Tant en l'àmbit econòmic i ambiental, crec que suposa un gran canvi. En l'àmbit social, pot ser que algunes persones hi tinguin complicacions (la gent gran per exemple), tot i això, crec que és qüestió de temps que es faci el canvi a tiquet digital, i tal com s'explica en la introducció, ja que molts negocis ja han començat a fer aquesta migració.

Per part del hardware, crec que el cost es pot abaratir més. En aquest projecte s'ha utilitzat la *RaspberryPi 3* perquè ja la tenia comprada des de fa temps, i el seu cost és de 35€. Aquesta, es podria substituir per una *RaspberryPi Zero*, la qual té un cost de 5.5€.

Per la part del software, crec que ha estat molt encertat la utilització de *sockets* per la comunicació entre el frontend i el backend. Normalment, les aplicacions web, criden al backend per agafar la informació que es demana, però en aquest cas, el frontend està en tot moment esperant que li arribi la informació del backend a través del *socket*.

El primer objectiu queda assolit amb la realització del projecte, ja que es presenta una solució fàcil d'implantar i que és paperless.

11.1 Futurs projectes

Alguns dels futurs projectes que es podrien fer són els següents:

1. Crear una aplicació mòbil, que mantingui tot l'historial dels tiquets que s'han descarregat a partir del codi QR.

2. Tal i com es diu a les conclusions, fer el mateix projecte, pero muntat damunt d'un hardware encara més barat que la *RaspberryPi 3*.

Bibliografia

[1] *¿Qué hacer con el 'ticket' de compra y para qué sirve?* Accedit el: 24-02-2021. URL: https://www.bbva.com/es/ticket-compra-sirve/ (v. la pàg. 1).

[2] *Cero papel en las empresas: un gesto por el medioambiente*. Accedit el: 23-02-2021. URL: https://www.lavanguardia.com/natural/20181219/453638207354/cero-papel-empresas-gesto-medioambiente.html (v. la pàg. 1).

[3] *Skip the Slip Report: Environmental Costs & Human Health Risks of Paper Receipts*. URL: https://www.greenamerica.org/report-STS (v. la pàg. 2).

[4] *Bisfenol A*. Accedit el: 04-03-2021. URL: https://echa.europa.eu/es/hot-topics/bisphenol-a (v. la pàg. 3).

[5] *Bisfenol A*. Accedit el: 04-03-2021. URL: https://www.sciencedirect.com/science/article/pii/S0269749120320947 (v. la pàg. 3).

[6] *TICKET DIGITAL CON LIDL PLUS*. Accedit el: 26-02-2021. URL: https://www.lidl.es/es/lidl-plus-ticket-digital/s2862 (v. la pàg. 5).

[7] *TICKET DIGITAL: DECATHLON*. Accedit el: 26-02-2021. URL: https://medioambiente.decathlon.es/sostenibilidad/ticket-digital/ (v. la pàg. 5).

[8] *Mango apuesta por la tecnología online impulsando el ticket digital e implantando el stock integrado en sus tiendas*. Accedit el: 26-02-2021. URL: https://www.trendencias.com/marcas/mango-apuesta-tecnologia-impulsando-ticket-digital-e-implantando-stock-integrado-sus-tiendas (v. la pàg. 5).

[9] *No Mas Tickets*. Accedit el: 26-02-2021. URL: http://www.nomastickets.com/ (v. la pàg. 5).

[10] *Google Cloud*. Accedit el: 30-04-2021. URL: https://cloud.google.com/ (v. la pàg. 17).

[11] *Google Drive*. Accedit el: 30-04-2021. URL: https://drive.google.com/ (v. la pàg. 17).

[12] *CUPS-PDF*. Accedit el: 27-04-2021. URL: https://www.cups-pdf.de/ (v. la pàg. 18).

[13] *GitHub*. Accedit el: 16-03-2021. URL: https://github.com/ (v. la pàg. 19).

[14] *Python*. Accedit el: 16-03-2021. URL: https://www.python.org/ (v. la pàg. 19).

[15] *Flask*. Accedit el: 16-03-2021. URL: https://flask.palletsprojects.com/en/2.0.x/ (v. la pàg. 20).

[16] *Angular*. Accedit el: 16-03-2021. URL: https://angular.io (v. la pàg. 21).

[17] *TypeScript*. Accedit el: 18-03-2021. URL: https://www.typescriptlang.org/ (v. la pàg. 22).

[18]*Node*. Accedit el: 18-03-2021. URL: https://nodejs.org/en/ (v. la pàg. 22).

[19]*Npm*. Accedit el: 18-03-2021. URL: https://www.npmjs.com/ (v. la pàg. 22).

[20]*Socket IO*. Accedit el: 30-04-2021. URL: https://socket.io/ (v. la pàg. 23).

[21]*Docker*. Accedit el: 19-05-2021. URL: https://www.docker.com/ (v. la pàg. 23).

[22]*RaspberryPi 3*. Accedit el: 30-04-2021. URL: https://www.kubii.es/home/1628-nouveau-raspberry-pi-3-modele-b-1-gb-640522710850.html?src=raspberrypi (v. la pàg. 76).

[23]*Google Drive Price*. Accedit el: 30-04-2021. URL: https://one.google.com/about/plans (v. la pàg. 79).