OPTIMITZACIÓ DE SERVEIS MICRO FRONT-END

Construcció i comunicació de micro serveis

Autor: Aleix Costa Recto

Ponent: Silvia Llorente Viejo

Director: Oriol Hernan Galobart

25-01-22

Universitat Politècnica de Catalunya
Màster en Enginyeria Informàtica

Índex

Índex d'il·lustracions

Índex de Taules

Resum

Aquest projecte de final de fi de Màster consisteix en l'optimització de serveis Front-End[1]. Estarien compostos pel concepte recent de Micro Front-End's[2], a partir d'ara MFE. Aquests són fragments del Front-End que treballen independentment per respondre a una pàgina web. Per fer-ho, s'utilitzaran diferents eines que permetin la creació d'aquests d'una manera fàcil, ràpida i dinàmica.

A més a més, s'implementarà una arquitectura Client-Servidor on posarem en pràctica tots els conceptes apresos en la fase d'investigació, poden així crear un sistema basat en MFE.

Finalment, podrem observar les funcionalitats que podem donar a cadascun d'aquests serveis tant de manera aïllada com de manera conjunta.

Resumen

Este proyecto de final de fin de Máster consiste en la optimización de servicios Front-End. Estarían compuestos por el reciente concepto de los Micro Front End[2], a partir de ahora MFE. Estos son fragmentos del Front-End que trabajan independientemente para responder a una página web. Para hacerlo, se usarán diferentes herramientas que permitan la creación de estos de una manera fácil, rápida y dinámica.

Además, se implementará una arquitectura Cliente-Servidor donde pondremos en práctica todos los conceptos aprendidos en la fase de investigación, pudiendo así crear un sistema basado en MFEs.

Finalmente, podremos observar las funcionalidades que le podemos dar a cada uno de estos servicios ya sea de manera independiente o en conjuntos con otros.

Abstract

This project consist of the optimization of Front-End services. These services are composed by the innovative concept of Micro Front End[2], hereinafter MFE. These are Front-End fragments that run independently to satisfy a whole web. To ease the process, we are going to use tools that make the use of MFE fast, dinamic and confortable to programmers.

In addition, a Client-Server architecture will be implemented where we will put into practice all the concepts learned in the research phase, creating a system based on MFEs.

Finally we will be able to observe the functionalities that we can give to each of these services either independently or in conjunction with others.

1 Definició de l’abast i contextualització

1.1 Context

1.1.1 Introducció

Aquest projecte és un Treball de Fi de Màster en el Màster d'Enginyeria Informàtica. Es realitzarà durant les pràctiques extracurriculars en l'empresa NTTData[3], treballant en un dels seus departaments anomenat DEx (Digital Experience) els quals s'encarreguen de canals digitals com serien portals web, aplicacions Mobile i un llarg etcètera.

Els micro serveis[4] són tanmateix una manera de programar el desenvolupament software com un estil d'arquitectura. Amb els micro serveis, les aplicacions es divideixen en els seus elements més petits i independents entre si. A diferència del model d'arquitectura monolític tradicional de les aplicacions, en el que tot es compila en una sola peça, els micro serveis son element independents que funcionen en conjunt per portar a terme les mateixes funcionalitats. Cada un d'aquests elements són un micro servei o un MFE.

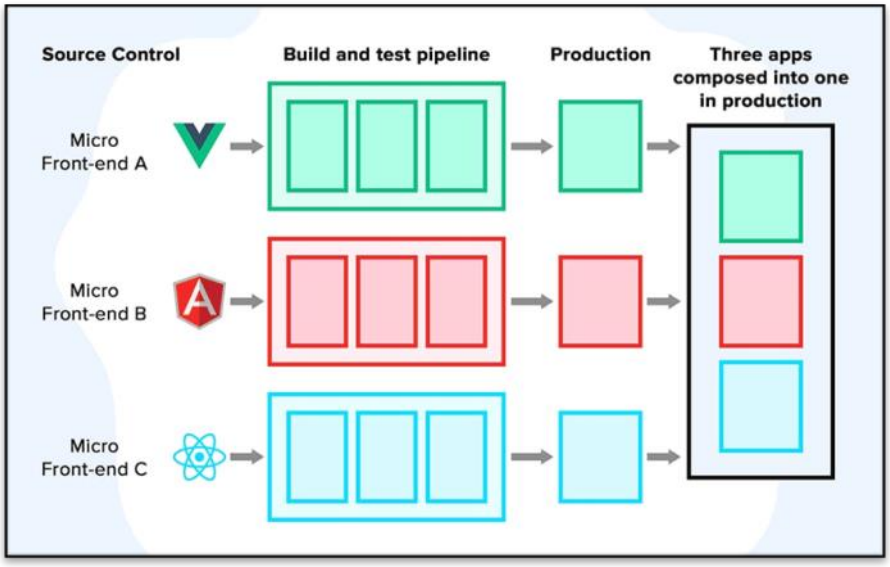


Figura 1: Esquema d'una arquitectura MFE Font:cygnismedia.com/

Aquest enfocament de desenvolupament de software valora el nivell de detall, la senzillesa i la capacitat per compartir un procés similar en vàries aplicacions. És un element fonamental de l'optimització del desenvolupament d'aplicacions pensant en un model compartit a la intranet. És a dir, podem utilitzar la mateixa aplicació replicada a diferents entorns webs!

L'objectiu d'aquest projecte és conèixer més a fons aquesta arquitectura no-monolítica, investigar quins mètodes de creació de MFE existeixen i fer les proves corresponents per poder acabar construint un sistema on tots els seus elements són independents i existeixi una connexió entre ells. Tractarem de millorar el punt on la creació d'aquests components sigui molt més fàcil, eficient i ràpida. Això s'aconsegueix usant una sèrie de

frameworks[1]que permetin la creació d'aquests, la seva gestió i ofereixin diferents funcionalitats internes per la seva comunicació. Finalment, podrem simular un entorn usant un sistema de micro serveis on cada equip podrà treballar cada un dels seus components de manera independent i paral·lelament.

1.1.2 Actors implicats

El projecte va dirigit a qualsevol usuari de Front-End, és a dir, desenvolupadors Web que utilitzin la tecnologia del projecte.

Autor del projecte: El desenvolupador de tot el projecte és l'autor d'aquest, és l'encarregat a assolir tots els objectius proposats, la documentació d'aquests, fer la prova de concepte pertinent i finalment presentar-lo.

Ponent del projecte: En aquest cas Silvia Llorente Viejo, és l'encarregada de supervisar els punts claus el projecte, resoldre dubtes i aportar els recursos necessaris.

Supervisor del TFG: Oriol Hernan Galobart ha estat l'encarregat de supervisar en l'àmbit tècnic el projecte, a més a més, ha estat el principal impulsor d'adaptar el projecte a les necessitats requerides, supervisar el compliment del calendari, veure com els objectius s'anaven complint i corregir en tot moment l'enfocament del projecte de manera dinàmica.

Equip de DEX: Ha estat l'equip encarregat de subministrar les tecnologies requerides, resoldre i gestionar correctament les peticions de configuració i aprovisionament de la infraestructura del projecte. L'objectiu del projecte els hi ajuda directament a les peticions que el client els hi fa, en aquest cas, Allianz[5]. Són els principals beneficiaris del projecte.

Usuaris finals: A més a més dels clients de l'equip de DEX, l'eina implementada i configurada és de codi obert, per tant, tothom podrà fer-ne ús si arriba a les seves mans.

1.2 Abast del projecte

Com ja hem dit anteriorment, aquest projecte té com a objectiu la investigació dels micro serveis i la creació d'aquests utilitzant tecnologies específiques per a poder satisfer l'eficiència, les facilitats i la rapidesa a l'hora de treballar amb el client i poder tenir un entorn de treball còmode i una interfície d'usuari agradable per a l'usuari final.

1.2.1 Objectius

Aquest projecte és un més del conjunt que forma l'equip de DEX. Durant el cicle de vida del departament, s'han trobat amb casos que poden facilitar, fer més eficients i amenitzar els processos escaients per a creacions d'entorn Web. Així doncs els objectius específics i generals d'aquest projecte, són:

[1] Eines que et faciliten la programació en qualsevol àmbit. Estan explicats en profunditat a la secció 5.2

- Objectius generals:

  - ✓ Investigació sobre l'arquitectura de micro serveis, creació d'aquesta, comunicació dels diferents MFE i creació d'un entorn web amb varis micro serveis.
- Objectius específics:

  - ✓ Anàlisi general MFE.
  - ✓ Investigació d'un sistema efectiu per integrar MFE
  - ✓ Prova de concepte amb els diferents sistemes.
  - ✓ Investigació de com comunicar aquests micro serveis.
  - ✓ Prova de concepte amb els diferents mètodes de comunicació als diferents sistemes estudiats.
  - ✓ Construcció d'una aplicació que contingui diversos MFE.
  - ✓ Creació d'un Back-End[6] i d'una connexió entre Front i Back.
  - ✓ Configuració general d'aquest sistema.

1.2.2 Possibles obstacles

Durant el projecte poden sorgir certes incerteses sobre aquest mateix i algunes dificultats que facin que modifiquin la metodologia, la planificació temporal i altres característiques envers el projecte, algunes d'elles poden ser:

- Desactualització d'algun framework a l'hora de fer el primer test. (P.ex: No es pot utilitzar a Windows 10)
- Incompatibilitat d'alguna tecnologia de comunicació amb algun dels framework estudiats. (P.ex: Alguna eina de les estudiades amb qualsevol framework)
- Necessitat de formació amb alguns dels llenguatges de fronts per a la creació d'alguns micro serveis. (Frameworks especficis en Front
- Limitacions que no contemplàvem d'alguna de les tecnologies del marc pràctic.
- El desplegament, el codi i el mètode de configuració de cada MFE pot ser diferent, així que implica un esforç addicional de comprensió.

1.3 Metodologia

S'ha elegit la metodologia basada en objectius SMART[2][6] utilitzant la tecnologia de Trello, un software d'administració de projectes amb interfície on aquesta et proporciona una solució completa per la gestió de projectes i treballs.

S'ha organitzat de la manera que el projecte es controli per objectius i, quan s'han complert uns quants o han conclòs amb alguna de les fases troncals, el supervisor del projecte realitzarà els compliments dels objectius, revisar-los, corregir-los i establir la següent fase. Tot això amb les corresponents fites establertes al principi de la metodologia.

S'utilitzarà GitHub[7] i Drive[8]per al control de codi i documentació respectivament i per a gestionar els canvis i modificacions.

Per a tota la gestió de codi, el desenvolupament, el testing dels MFE i per a la integració d'aquests s'utilitzarà Visual Studio Code[9] com a editor de text. Per a la creació de Back-End s'ha proposat l'ús de json-server[10], Swagger[11] o MockApi[12].

[2] SMART: Specific, Measurable, Achievable, Realistic i Timely, metodologia per establir objectius en un projecte

2 Planificació temporal

El projecte, des d'un primer moment, té l'objectiu de realitzar-se en unes 18-20 setmanes, començant a comptar la primera setmana de setembre i intentar acabar-lo just a inicis de gener, les hores destinades al projecte estan al voltant de 600-700. Tota aquesta planificació no inclou, la gestió del projecte en si, com el resum, els objectius, la motivació, l'estat de l'art, la metodologia, etc. Per tant, en aquestes dates només contemplem la informació interna del TFM (Marc Teòric i Marc Pràctic).

2.1 Descripció de les tasques

2.1.1 Anàlisi general dels MFEs

A l'inici del projecte caldrà una iniciació sobre els conceptes que es tractaran durant tot el treball. S'haurà de familiaritzar en conceptes com: Arquitectura Monolítica, Microserveis, Front-End Frameworks i altra informació addicional com ara perquè serveixen aquestes arquitectures, com les podem crear, on s'utilitzen...

Un cop feta la primera part teòrica del projecte, accedirem a una demo pràctica ja creada per observar com es veuen ambdues arquitectures a nivell de codi. Per la part pràctica, voldrem aproximar-nos a una arquitectura visual com la que mostrem a continuació:

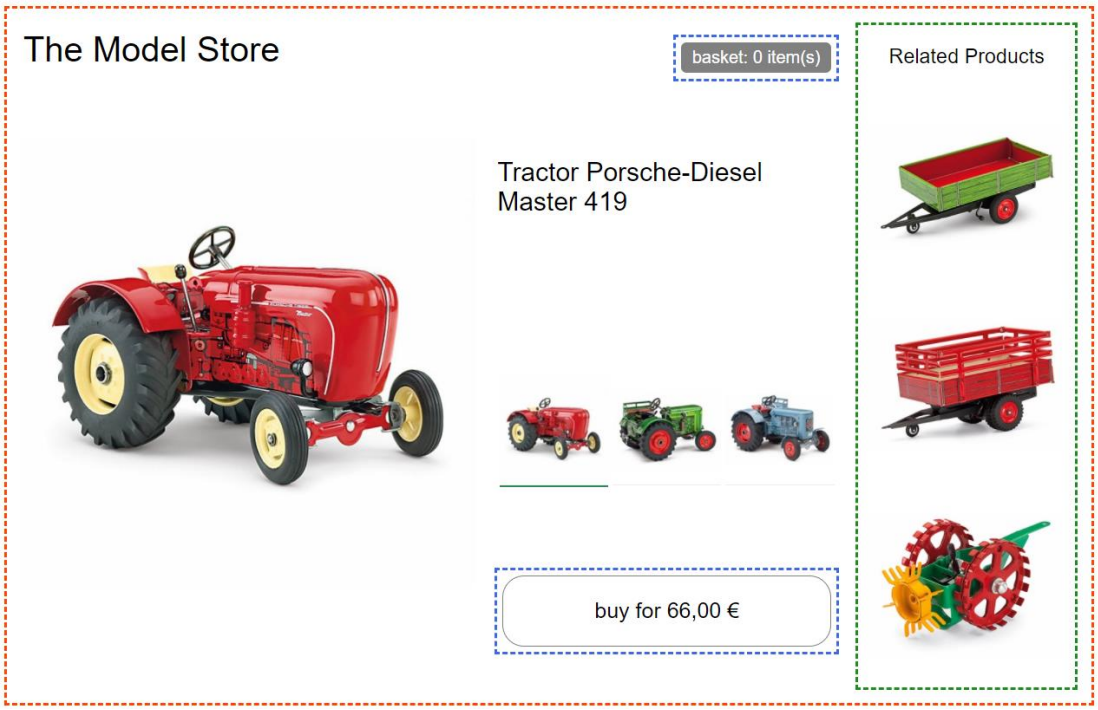


Figura 2: Disseny d'un MFE en una pàgina web Font: micro-frontends.org

Tindrà una durada aproximada de 60 h entre tota la recerca, documentació llegida i proves per entrar en contacte amb aquest sistema.

2.1.2 Investigació d'un framework per l'integració dels MFEs

Un cop tenint tots els conceptes apresos i documentació llegida corresponent, haurem de fer una anàlisi exhaustiva de les tecnologies a incorporar per a saber quins tindran millor rendiment per a la construcció d'un portal web.

S'haurà de fer una anàlisi detallada dividit amb 3 camps: Seguretat, Maduresa i Característiques.

Dins del camp de la seguretat ens centrarem a estudiar el nivell de seguretat addicional que ens aporta aquesta tecnologia dins del gestor, tractarem termes com: eines antiatacs, anàlisi de vulnerabilitats o la gestió de les dades privades que fa cada una de les tecnologies.

Dins del camp de la maduresa ens centrarem a veure quina trajectòria porten en la història aquests frameworks i quin està millor cuidat i mantingut, tractarem termes com: documentació de cada tecnologia, la seva comunitat de desenvolupadors, els suport del fabricant de cadascun i el seu llicenciament i costos per a utilitzar-los.

I finalment, per les característiques estudiarem i analitzarem termes com ara: ús d'estils de cada framework, la seva escalabilitat i adaptabilitat amb altres tecnologies, l'ús de les plantilles i la seva compatibilitat Cross-Browser (entre els diferents navegadors).

Tindrà una durada aproximada de 90h per a fer tota la documentació de les característiques.

2.1.3 Investigació de com comunicarem cada MFE

Una vegada estudiats cada un dels frameworks per a la creació de micro serveis i havent fet el corresponent anàlisi exhaustiva, continuarem amb la comunicació entre MFE.

En aquest punt mirarem quines maneres existeixen de comunicar "projectes independents" que romanen en una mateixa pàgina. Explicarem com poder-ho implementar en cadascun dels frameworks estudiats. Farem un estudi de diverses maneres de comunicació i portarem a la pràctica una o dues formes, les que més interessants ens semblin.

Aquesta fase constarà d'unes 70h entre la investigació de totes les maneres, la comprensió d'aquestes i l'anàlisi per acabar triant alguna d'elles.

2.1.4 Prova de concepte dels frameworks i dels mètodes de comunicació

Aquest apartat conté dos dels objectius prèviament proposats. Per una banda, farem la prova del frameworks analitzats exhaustivament per veure quines sensacions transmeten a l'usuari. Veurem també les dificultats que ens poden oferir i com es desenvolupen de manera pràctica. Farem un petit exemple amb cadascun d'ells, crear així alguns components molt bàsics per veure tot el potencial que poden tenir. No entrarem al detall, simplement que ens ofereixen a grans trets.

Per una altra banda, provarem aquests frameworks amb els mètodes de comunicació triats. Veurem si tenen alguna incompatibilitat o es necessita alguna eina especial per al

desenvolupament de la prova conjunta. Aquesta fase comportarà també la creació de components que tinguin la capacitat de comunicar-se entre si.

Aquesta fase constarà d'unes 120h entre la instal·lació de totes les eines, la implementació de tots els mètodes i la creació dels components que es comunicaran.

2.1.5 Creació d'una aplicació que contingui MFE

A partir d'aquí entrem en les fases pràctiques del projecte. Voldrem crear una espècie de portal web utilitzant les tecnologies estudiades i les eines de comunicació analitzades. Per a fer-ho primer disposarem de la creació de tots els webs components necessaris pel Front-End de la nostra aplicació. Aquests seran basats en components pre-elaborats del client del nostre departament: Allianz.

Amb aquesta aplicació podrem simular com podem treballar de manera independent amb cadascun dels components i simularem com treballen 2 o 3 equips de manera conjunta per la creació de cadascun dels micro serveis.

Aquesta part constarà de 144h entre tota la creació del sistema.

2.1.6 Creació d'un back-end configurable

Aquest punt pot variar durant la part pràctica del projecte. La idea d'aquest apartat és establir comunicacions entre els components mitjançant Back-End. S'haurà de tenir clar quina informació es comunicarà només amb un component concret o també es voldrà aquesta informació per la resta. També disposarem d'aquesta comunicació utilitzant Front-End amb tecnologies que estudiarem més endavant.

No s'ha tingut mai l'experiència de crear un Back-End des de zero, s'haurà d'investigar quina tecnologia és millor i de com s'estructurarà. Aproximo unes 120h.

Tasques	Hores	Dates	Risc
Anàlisi general dels MFEs.	60	06/07/21-20/07/21	Baix
Investigació d'un framework per l'integració dels MFE.	90	21/07/21-20/08/21	Baix
Investigació de com comunicarem cada MFE.	70	23/08/21-17/09/21	Mitjà
Prova de concepte dels frameworks i dels mètodes de comunicació.	120	20/09/20-22/10/20	Mitja-Alt
Creació d'una aplicació que contingui MFE.	144	25/10/21-30/11/21	Alt
Creació d'un back-end configurable.	120	1/12/21-24/12/21	Molt Alt

Documentació i manual d'usuari de tota la infraestructura creada.	120	27/12/21-17/01/22	Mitjà-Alt
Total	724	06/09/21-17/01/22	

Taula 1: Resum tasques a fer

2.1.7 Documentació i manual d'usuari de tota la infraestructura creada

Ja com a finalització del projecte, s'haurà de documentar totes les característiques que conté cada component (en un principi es voldrà dissenyar com a mínim 3 components dinàmics), el seu manual d'usuari, és a dir, totes les funcionalitats que conté cada un d'ells i finalment intentarem fer l'aplicació d'un cas real amb el client, que això s'encarregarà de gestionar-ho l'equip de DEX amb el client ALLIANZ. A més es mostrarà com s'ha fet l'arquitectura Front-End amb els micro serveis i el corresponent Back-End amb la compartició d'informació amb cada component. Aproximo una durada de 120h.

Finalment es prepararà la defensa i es documentarà tot el projecte de forma adient.

2.2 Recursos utilitzats

Ho dividim entre: Recursos Humans i Recursos Tècnics.

Pel que fa a els recursos tècnics farem menció a les tecnologies.

1. Visual Studio Code: Un editor de codi per a fer la programació de tots els components.
2. Node.js[13]: Entorn d'execució JavaScript per a la realització de components utilitzant els frameworks que volem.
3. MFE: El concepte/arquitectura sobre el que treballarem en tot el projecte.
4. GitHub: Entorn per a guardar codi i compartir-lo amb l'equip de DEX.
5. Ordinador DELL Core i5 amb Windowns, màquina que s'ha fet servir per a tot el desenvolupament del projecte.
6. Trello[14], eina per a la gestió del temps.

Pel que fa a els recursos humans farem menció a la gent implicada:

1. Autor del projecte: El desenvolupador de tot el projecte és l'autor d'aquest, és l'encarregat de dur a terme la seva implementació.
2. Ponent del projecte: En aquest cas, Silvia Llorente Viejo és l'encarregada de supervisar en punts claus el projecte, resoldre dubtes i aportar els recursos necessaris.
3. Director del TFM: Oriol Hernan Galobart ha estat l'encarregat de supervisar a nivell tècnic el projecte, a més a més, ha estat el principal impulsor d'adaptar el projecte a les necessitats requerides, supervisar el compliment del calendari, veure

com els objectius s'anaven complint i corregir en tot moment l'enfocament del projecte de manera dinàmica.

4. Equip de DEX: Ha estat l'equip encarregat de subministrar les tecnologies requerides, resoldre i gestionar correctament les peticions de configuració i aprovisionament de la infraestructura del projecte.

2.3 Estimacions i Gantt

El diagrama de Gantt proposat inicialment, no fa justícia a la planificació de totes les tasques que realment s'ha portat a terme finalment.

Els 4 primers punts de la Taula 1 mostrada en l'apartat superior s'han assolit satisfactòriament sense cap tipus de problema, però, a partir del 5é hi ha hagut modificacions tant en l'àmbit de planificació temporal com a objectius esperats.

Un cop teníem els frameworks escollits i els mètodes de comunicació entre els MFE treballats i estudiats vam procedir a la integració de varis MFE amb varis frameworks (Fent referencia al punt de: Creació d'una aplicació que contingui MFE) utilitzant Module Federation[15], un dels frameworks estudiats. L'elecció d'aquest és causada per unes puntuacions extretes tant en l'àmbit teòric com pràctic i també ens vam guiar pel que tenia més suport per la comunitat.

Per altra banda, l'aplicació que volíem crear havia de contenir com a mínim 2 frameworks diferents, això és perquè, per definició, els MFEs tenen el potencial de poder barrejar vàries versions, varis frameworks de front en una mateixa aplicació web. A més a més, el que volíem aconseguir com la cúspide d'aprofitament de tot el potencial era que en una mateixa pantalla d'una pàgina web puguem diferenciar els diferents MFE i que cadascun d'ells estiguessin programats amb un framework de front diferent.

Un cop ens trobàvem en aquest punt vam tenir una infinitat de problemes amb el fet de poder crear connexió amb dues aplicacions Angular[16] utilitzant Module Federation. Vam provar moltes maneres de poder fer aquesta connexió. Vam començar usant una aplicació contenidora tipus angular i un MFE tipus Angular també i l'únic que vam obtenir al final de 30 dies de proves, va ser, mitjançant Routing i l'aixecament d'un servidor temporal, aconseguir aquesta connexió, però quan intentàvem sortir de l'encaminament, no vam poder executar-ho en condicions. Evidentment, també vam provar diferents arquitectures (Contenidor amb React[17] i MFE amb Angular, Contenidor amb Angular i MFE amb React i varies més).

Al final vam arribar a la conclusió que Angular és un ecosistema massa propi i amb Module Federation no hi ha els suports esperats, ja que és una tecnologia que sense exagerar fa 1 any que te mètodes funcionals. A més a més Angular cada 6 mesos treu una release nova amb difunts mecanismes que poden afectar amb la interacció amb Module Federation.

Tot això en parlarem molt més en detall a la memòria final del projecte.

Evidentment no comptàvem amb aquest contratemps hi hem suprimit gairebé l'apart de "Creació d'un Back-End Configurable". Ho hem simplificat molt perquè la nostra aplicació al final contingui les següents característiques:

- ✓ Dos o més MFEs.
- ✓ Comunicació amb un Back-End dinàmic
- ✓ Routing en algun dels components
- ✓ Varis frameworks en una pàgina web
- ✓ Visualització d'aquests en una mateixa pantalla.

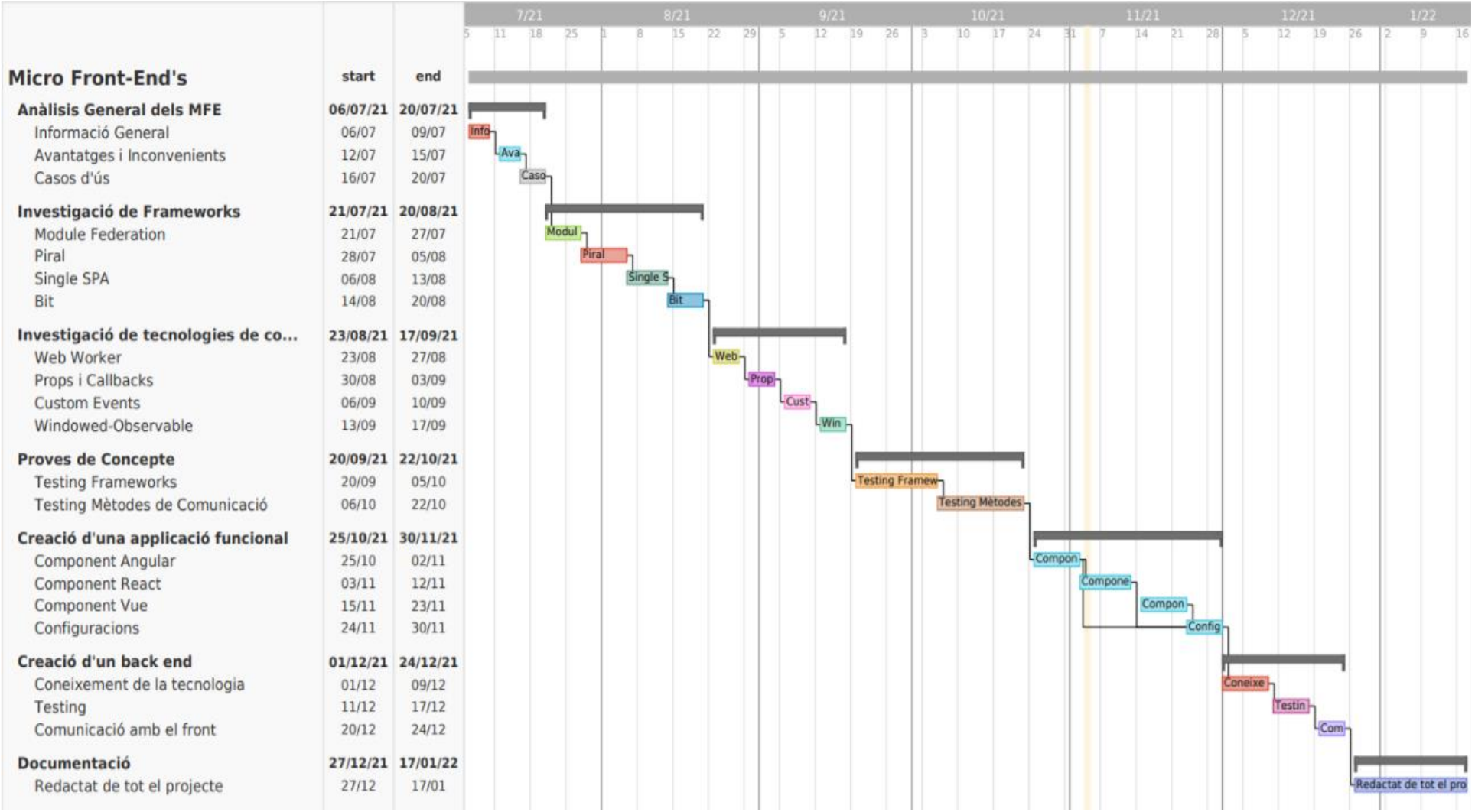


Figura 3: Diagrama de Gantt Font: TeamGantt.com

3 Gestió econòmica i sostenibilitat

3.1 Identificació de costos

Tot seguit parlarem dels diferents costos que s'han de tenir en compte i els costos aproximats de cada element per a la realització del projecte.

Ho dividirem en recursos humans i recursos materials, incloent-hi Software, Hardware, etc.

3.1.1 Recursos Humans

Tindrem en compte fins a 3 vessants en tot l'àmbit dels recursos humans, aquests són:

- **Autor/Desenvolupador del Projecte**: Encarregat de la correcta implementació del projecte, desenvolupar codi i fer les corresponents proves de concepte de totes les tecnologies. En aquest cas només hi haurà un desenvolupador que acomplirà totes les tasques explicades en el Gantt. Vam dir que el projecte tenia una durada d'unes 700h sense tenir en compte la formació prèvia i el reconeixement de l'Autor amb l'empresa, ens surt a un cost total de 9100€ a 13€/h.
- **Equip DEX**: Encarregat de subministrar les tecnologies requerides, resoldre i gestionar correctament les peticions de configuració i aprovisionament de la infraestructura del projecte. L'objectiu del projecte els hi ajuda directament a les peticions que el client els hi fa, en aquest cas, Allianz. Es compta amb un equip de 8 persones properes amb un cost de 30€/h fent unes 30h cadascú, acaba sortint a 7200€.
- **Gestor del projecte**: Encarregat del correcte desenvolupament del projecte i el principal responsable de la direcció. Escala decisions i assigna recursos. El gestor dedicarà unes 30h en aquest projecte essent el seu cost total, a 45€/h , 1350€.

En total el pressupost de recursos humans serà de: 15.550€.

Recurs	**Cost**
Autor	9.100€
Equip DEX	7.200€
Gestor Projecte	1350€
Total	17.650€

Taula 2: Resum recursos humans

3.1.2 Recursos Materials

Suposarem el cost dels components software i hardware que s'han inclòs en aquest projecte:

- **Portàtil d'empresa**: S'utilitza una unitat. El seu cost és d'uns 1400€. La seva vida útil són 34800 hores i, per tant, el seu preu per hora és d'uns 0.0427€. En aquest projecte s'utilitza durant totes les fases del projecte, per tant, s'utilitza durant 700h. 30,0€ serà el cost destinat aproximat.

- **Software**: Windows. El sistema operatiu de cada entorn de treball també s'ha de comptabilitzar com un cost. En aquest cas el cost és de 100€ per sistema operatiu, com només s'utilitza un portàtil és el cost final.
- **Llicència Microsoft Office**: Microsoft és totalment necessari per a la realització d'aquest projecte, o com a mínim és la que utilitza l'empresa per a portat tots els spreadsheets, presentacions i documentacions vàries. El seu cost és de 8€/mes, com que el projecte dura uns 4-5 mesos el cost serà de 40€.
- **Despeses logístiques i generals**: Existeix tot el cost de l'equip de treball utilitzat pel desenvolupador del projecte, aquest es compon de la zona, teclat addicional, pantalla addicional i demés cablejat de l'empresa, surt a uns 1500€. El lloc de treball també el tenim en compte i acaba sortint a uns 200€ al mes que es tradueix a 1000€ totals.

El total del pressupost dels recursos materials és de: 2670,0€.

Recurs	**Cost**
Portàtil Empresa	30€.
Software	100€
Llicència Microsoft Office	40€
Despeses Logístiques	2.500€
Total	2.670€

Taula 3: Resum recursos materials

3.1.3 Possibles imprevistos

Hem de valorar que aquest càlcul és només una estimació del que pot arribar a costar tot el projecte en si, hem de preveure canvis per tenir prou potencial econòmic i tenir marge de maniobra a aquests possibles canvis.

Recursos humans: Per aquesta banda podríem necessitar més hores de reunions amb el tutor del projecte per a possibles canvis en la metodologia, o que l'equip de DEX ens dediques més hores en l'elaboració d'aquest. Augmentarem en 1.500€ el pressupost total de recursos humans.

Recursos materials: Com que només tenim costos fitxes i software que no pot fallar no existeix la possibilitat que haguem de fer una estimació de diners extres de l'equip material, però sí que és possible que el portàtil d'empresa es malmeti i faci falta reemplaçar-lo i suposaria un cost addicional de 1000€.

Imprevistos	**Cost**
Recursos Humans	1.500€
Recursos Materials	1.000€
Total	2.500€

Taula 4: Resum taula imprevistos

3.1.4 Pressupost final

Taula resum amb els costos subdividits.

Recurs	**Cost**
Recursos Humans	17.650€
Recursos Materials	2.670€
Imprevistos	2.500€
Total	22.820€

Taula 5: Resum de costos final

3.1.5 Control de gestió

L'empresa on es desenvolupa aquest projecte utilitza 5 tecnologies principals per al control de gestió de les infraestructures, costos i demandes.

Jira(JIRA)[18] és una eina en línia per l'administració de tasques d'un projecte, els seguiments d'errors i la gestió operativa de projectes. És la més utilitzada a l'empresa i serà l'encarregada de portar els comptes de projecte.

Per la documentació de la informació dels projectes i la infraestructura utilitzem Confluence[19].

Pel versionat del codi s'utilitza GIT.

Per la construcció i el desplegament d'aplicacions s'utilitza Jenkins[20].

I per la imputació de les hores de l'equip s'utilitza OneERP[21] que és una eina pròpiament de l'empresa.

El cost total d'aquestes eines és 0€ ja que és un software totalment gratuït.

3.2 Informe de Sostenibilitat

3.2.1 Dimensió ambiental

L'impacte ambiental d'aquest projecte és la reutilització que li poden donar a tota la investigació proposada en un escenari futur. Al parlar d'arquitectures molt noves i poc treballades en l'actualitat, sí que a futur pot tenir un impacte més significatiu del que està suposant ara.

Encara que avui dia trobem varis articles, xerrades, conferències i algun projecte de MFE aplicat al món laboral on intervé un client, puc afirmar amb completa seguretat que no estem en el punt més àlgid per avaluar el projecte en l'àmbit de sostenibilitat ambiental. Ja que aquesta manera de fer feina encara es descobreix en fases de desenvolupament i no estaríem fent un estudi apropiat sobre aquest àmbit. Aquest s'hauria de fer en un període de 3 a 5 d'anys, on, en aquest marge de temps, sí que podrem aconseguir un impacte significatiu. Pel que fa a l'actual, és negligible.

De totes maneres, els MFE, com tot projecte de software, intervé un desgast d'energia i un cost computacional on al final tot això es tradueix amb "Co2" produït pels servidors

on realitzem connexions de la infraestructura i per la part del client on utilitzem l'energia del computador a on ens connectem a aquesta arquitectura.

3.2.2 Dimensió social

Des del punt de vista de la dimensió social, aquest projecte aporta un gran valor a persones que tenen coneixements en arquitectura de projectes en un primer nivell i també a les persones que estan acostumades a treballar amb tecnologies informàtiques similars a les que parlem en aquest projecte, però, sobre les que no estan en un àmbit pròxim podria ser un problema.

Sobre l'impacte personal del projecte ha sigut totalment positiu, ja que he tingut en compte situacions d'un projecte on amb anterioritat hagués ignorat per complet. He pogut aplicar metodologies apreses en el Màster així com coneixements tècnics al llarg de tot el projecte. El món dels MFE és una nova manera d'enfocar la part de la interfície gràfica d'un projecte informàtic.

Sobre la part de l'impacte social extern, si l'arquitectura dels MFE evoluciona i s'actua amb més profunditat causarà un gran impacte a la societat desenvolupadora d'aplicacions web. I no només als desenvolupadors d'un projecte, amb aquesta afirmació també s'inclouen gestors de projecte, proveïdors, consumidors (clients) o inclús tercers (usuaris indirectes).

Finalment, aquesta nova manera de treballar (amb MFE), farà al desenvolupador i als equips tenir una nova formació en l'àmbit del desenvolupament web, fent que en compte de seguir una formació horitzontal, on només es toca un àmbit del departament, es faci l'ús d'una vertical, obtenint així coneixement de tots els camps del departament on es trobi.

3.2.3 Dimensió econòmica

Sobre l'àmbit econòmic, hem realitzat una aproximació dels costos que pot tenir aquest projecte al punt 3.1 del TFM. Però a més a més dels costos exposats en l'anterior apartat, hem de valorar que ens estalviem amb les conclusions d'aquest projecte.

En una primera instància i parant d'un treball futur, tindrem la capacitat de fusionar el que ara són diversos departaments de Front-End, en un sol. Amb això em refereixo al fet que una empresa que tingui un equip amb coneixements de l'arquitectura MFE podrà treballar en projectes que utilitzin aquesta tecnologia, que engloba les altres de Front. Això en l'àmbit econòmic té un impacte en la infraestructura física d'una empresa, ja que reduïm diversos espais per només utilitzar-ne un.

Pel que fa a l'àmbit dels possibles ingressos d'aquest projecte, s'haurà de veure cap on tira aquesta metodologia de treball en el desenvolupament web. S'ha de tenir en compte els usos que pot tenir, l'èxit, el número d'implementacions o la quantitat de recursos que s'han d'invertir per dur a terme investigacions en aquest sector dels MFE. S'estima que en les futures investigacions es puguin beneficiar del treball realitzat en aquest projecte i així aportin un valor econòmic significatiu en els corresponents sectors.

4 Marc teòric

4.1 Micro Front-Ends (MFE)

Abans de començar a parlar d'aquest concepte haurem de veure primer que és un "Front-End" i un "Back-End" d'una aplicació. A més a més també hem de saber que un MFE no és res més que un micro servei, així doncs també introduirem aquest concepte.

4.1.1 Front-End i Back-End

En informàtica, el concepte de **Front-End** fa referència a la part del software que interactua directament amb els usuaris. En altres paraules, és la part visible, la que mostra el disseny, els continguts i la interfície gràfica. En canvi, el concepte del **Back-End** fa referència a la part invisible però fonamental en les aplicacions webs. És els processos que funcionen per darrere del que veiem i s'encarrega de tota la lògica perquè tot funcioni.

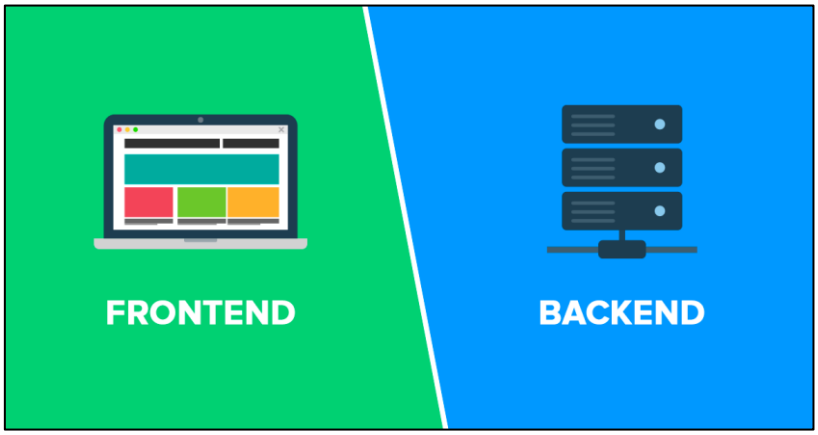


Figura 4: Front-End vs Back-End Font:codigocorrecto.com

4.1.2 Microserveis

Els micro serveis són un patró d'arquitectura a l'hora de programar amb software. Amb aquest patró les aplicacions es divideixen en els seus elements més petits i independents entre si. A diferència de les arquitectures monolítiques[3]en el que tot es compila en una sola peça, els micro serveis són elements independents que funcionen en conjunts per dur a terme els mateixos objectius.

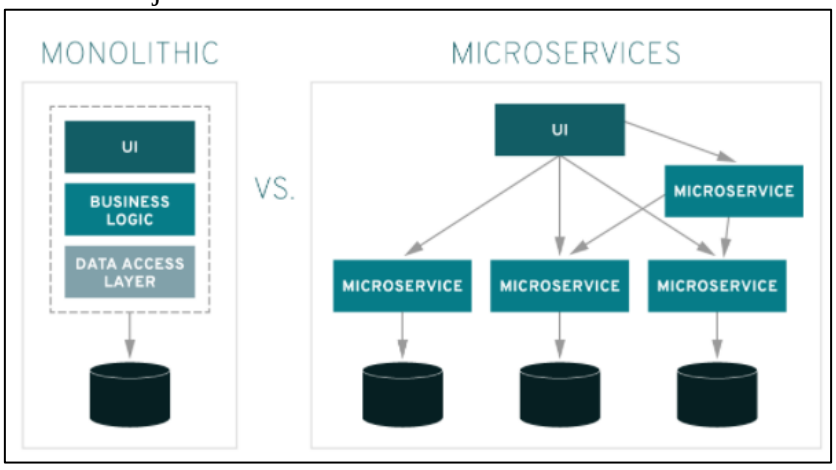


Figura 5: Arquitectura monolítica vs arquitectura MFE Font:geekflare.com/

[3]Model d'arquitectura de projectes en que existeix un sol projecte únic i tot es treballar sobre el mateix.

Un cop introduït aquests conceptes ja ens podem fer a la idea de que son els MFE. Però anem a explicar-ho tot des de l'arrel.

4.1.3 Concepte de MFE

En els últims anys, els micro serveis s'han fet molt populars. De fet, han aparegut varies vegades i de diferents maneres a TechRadar[4][22] des de 2016.

Els MFE són un conjunt de tècniques i estratègies per dur a terme aplicacions web amb un conjunt d'equips que puguin treballar de manera independent. És l'extensió de l'arquitectura de micro serveis al Front-End.

Cada equip s'encarrega d'una o més funcionalitats i les programa completament (barres de navegació, menús, botons, APIs[23] de pagament...). Després aquestes funcionalitats es componen per construir una aplicació final. El més ideal seria que les funcionalitats de cada equip no tinguessin massa dependències amb els altres equips perquè no hi hagi gaires problemes.

L'organització dels equips que treballen amb MFE no és per tecnologia(Front-End i Back-End), sinó que són equips transversals amb tots els perfils necessaris per desenvolupar una funcionalitat completa. La següent imatge aclareix tota l'explicació dels MFE.

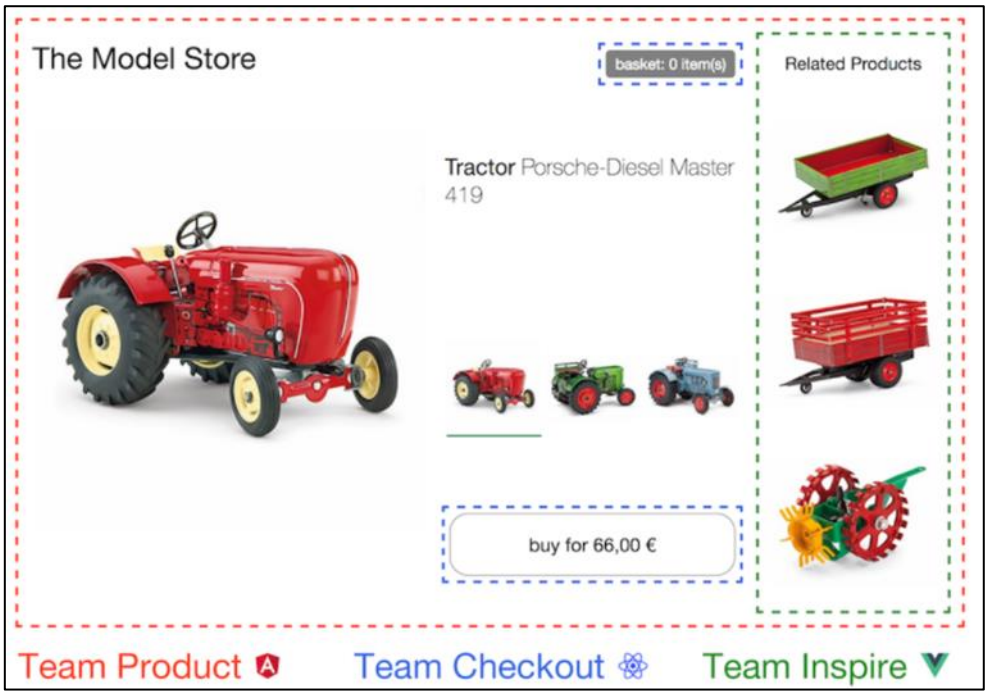


Figura 6: Visualització de MFE per equips : micro-frontends.org

[4] Pàgina de referencia mundial per a les tecnologies més punteres

4.1.4 Beneficis dels MFEs

Com hem vist, aquesta manera de treballar aporta millores i eficiència a l'hora de treballar amb equips, és per això que trobem diversos beneficis com són:

- Bases de codi més petites i sostenibles
- Organitzacions més escalables amb equips autònoms
- L'habilitat de millorar, actualitzar o sobreescriure parts del Front-End de manera més gradual que abans.

Però en el món dels MFE no tot són avantatges, també tenim alguns costos a patir: Algunes implementacions utilitzant aquesta arquitectura poden conduir a la duplicació de dependències, augmentant el número de bytes que l'usuari haurà d'acabar descarregant-se. A més a més, al tenir tanta autonomia pot provocat la fragmentació de la seva manera de treballar.

No obstant, acabem concloent que els beneficis acaben superant als costos. A continuació exposo cada una de les situacions beneficioses que poden tenir els MFE.

1. Actualitzacions incrementals

En una aplicació monolítica teníem el problema que cada vegada que volíem dur a terme algun canvi, una modificació o una actualització havíem de recompilar tota l'estructura des de zero. Amb l'arquitectura dels MFE, podem actualitzar lliurement de manera individual els nostres micro serveis. Si hi ha un canvi important al nostre marc principal, cada MFE pot actualitzar-se quan tingui sentit en comptes d'estar obligat a aturar tot el servei monolític i actualitzar-ho tot de cop.

2. Bases de codi senzilles i aïllades

El codi font de cada micro servei serà, per definició, molt mes petit que el codi font de un únic projecte monolític. Normalment aquest codi font serà molt més fàcil de treballar pels desenvolupadors. En particular, evitem la complexitat que prové de l'acoblament involuntari i inapropiat entre components que no haurien de tenir cap tipus de relació. Bàsicament, evitem erros de dependències quan volem provar alguna cosa que no té res a veure amb els erros que ens estan mostrant.

3. Desplegament independent

Tanmateix, com passa amb els micro serveis, la capacitat d'un desplegament independent és fonamental. Cada MFE hauria de tenir el seu canal d'entrega continua. Això consta de: Una construcció, un test i un desplegament final a producció. Hauríem de ser capaços de desplegar cada MFE sense haver de pensar en l'estat actual d'altres canals, com passa en l'arquitectura monolítica. Un MFE ha de ser independent a la resta de Front-End's del qual es composa una aplicació i només hauria de dependre d'ell mateix.

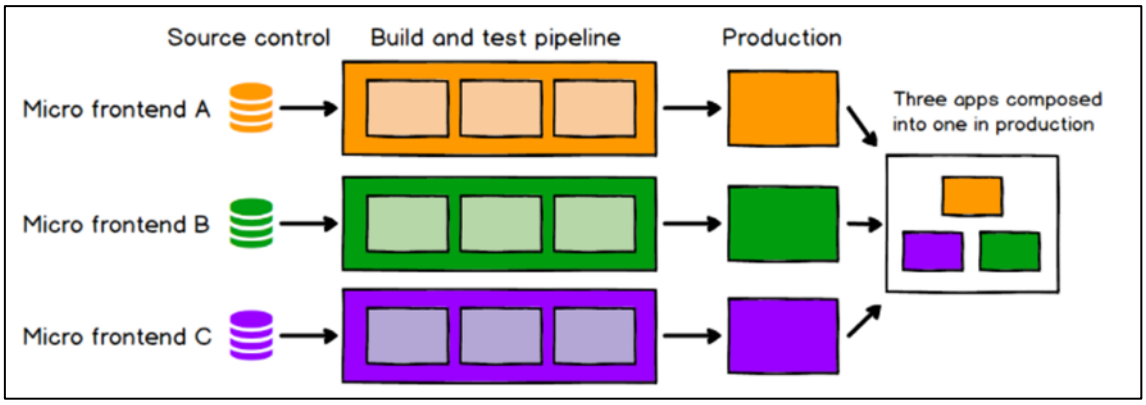


Figura 7: Il·lustració de com s'organitza una arquitectura MFE Font: medium.com/

4. Equips autònoms

Com ja hem comentat aquesta arquitectura ens permet tenir equips independents treballant sobre cada MFE. Això fa que l'estructura de treball horitzontal de les aplicacions monolítiques desaparegui i aparegui la vertical. Amb altres paraules, necessitem que els nostres equips tinguin la capacitat d'oferir valor als clients fent que hagin d'enfocar els seus coneixements a un àmbit empresarial en comptes capacitats tècniques exclusives. La següent imatge clarifica aquest punt:

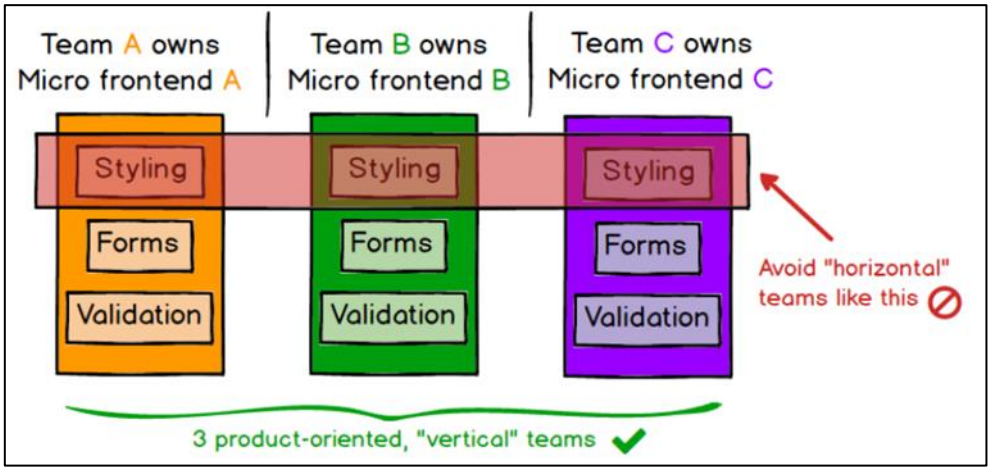


Figura 8: Organització horitzontal vs vertical en l'arquitectura MFE Font: /medium.com/

En resum, l'avantatge més gran d'un MFE és poder dividir les estructures més grans i enrevessades en peces més petites i maniobrables sempre i quan puguem definir de manera òptima les dependències de cada MFE

4.1.5 Aplicació d'una arquitectura MFE

Imaginem una pàgina web on els clients poden demanar menjar a domicili. En un entorn d'aquestes característiques tenim diverses parts a gestionar:

- Ha d'haver-hi una pàgina principal en la qual els clients puguin navegar i buscar ítems. Aquests s'han de poder buscar i filtrar-se per qualsevol número d'atributs, com pot ser el preu, la cuina o el que el client hagi demanat anteriorment.

- Cada restaurant ha de tenir la seva pròpia pàgina en el que es mostrin els elements del menú i que permeti al client que vol menjar, amb descomptes, ofertes i peticions especials.
- Els clients han de tenir una pàgina de perfil en la que puguin veure el seu historial de comandes, fer un seguiment de l'entrega i altres opcions de personalització.

Com podem apreciar, cada una d'aquestes parts són prou complexes com per tenir un equip dedicat a cadascuna d'elles i cada equip hauria de poder treballar de manera independent. Aquí és on entraria en joc l'arquitectura dels MFE.

4.1.6 Models d'integració dels MFE

Hi ha molts enfocaments que podrien dir-se MFE. En aquesta secció mostrarem alguns exemples i veurem avantatges i desavantatges d'aquests.

Normalment en una infraestructura de MFE hi ha un o més Front-End's per cada pàgina i una sola aplicació que fa de contenidora. Aquesta pot:

- Mostrar els elements comuns de la pàgina, com headers[5] i footers[6].
- Gestionar temes com l'autenticació i la navegació
- Recollir tots els MFE a la pagina i poder dir-li quan s'han de renderitzar cada un

A continuació veurem alguns exemples on aquesta estructura intervé:

1. Composició de fitxers HTML

Aquest model d'integració és conegut per molts, ja que no és massa innovador. Podem utilitzar HTML pur en el servidor a partir de múltiples plantilles o fragments. Suposem que tenim un index.html que conté els elements que volem renderitzar, però amb la característica que utilitzem un include d'un arxiu que és dinàmic. Podem adaptar aquesta variable al que ens interessa a partir de la URL que ens demani l'usuari i mostrar-li l'HTML que correspon.

Podem anomenar a això "MFE", ja que hem dividit el nostre codi de tal manera que cada peça representa un concepte de domini autònom que pot ser entregat per un equip independent.

[5] Part superior d'una pàgina web. És la capçalera on sovint existeixen les barres de menú horitzontals i altres menús.
[6] Part inferior d'una pàgina web. És el peu on sovint està la informació addicional, contactes i enllaços externs.

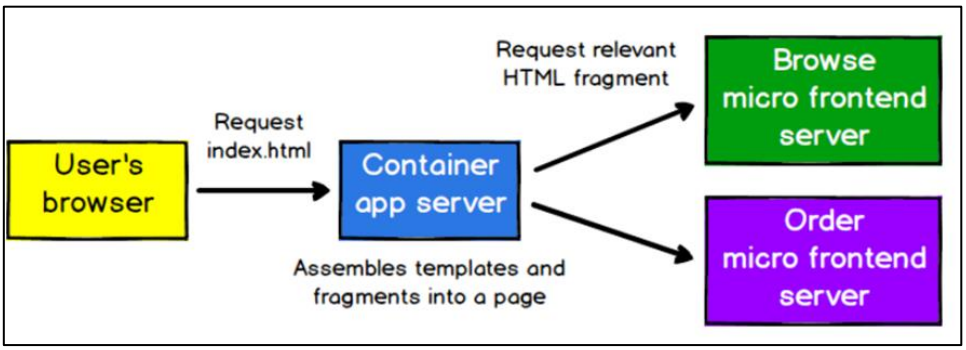


Figura 9: Exemple de composició a traves d'HTML Font: medium.com/

2. Integració en temps de compilació

Un altre enfocament podria ser el fet de publicar cada MFE com un paquet i fer que l'aplicació contenidora els inclogui tots com ho fa amb les dependències. Això és podria fer en un Package.json[7]. Aquesta manera és força valida però fer-ho així significa que hem de compilar cada MFE de manera independent cada cop que vulguem fer un canvi en els MFE de la contenidora.

D'aquí concloem que hauríem de trobar una manera d'integrar els nostres components en temps d'execució i no de compilació.

3. Integració en temps d'execució utilitzant iframes

Probablement, el model més simple i el més utilitzat. L'iframe pot ser utilitzat perquè, mitjançant una etiqueta <Script>, carregui diferents fragments de manera dinàmica i en temps d'execució. Els iframe són útils ja que faciliten la construcció d'una pagina a partir de subpàgines independents. A més a més també ofereixen un bon grau d'aïllament en termes d'estil i variables globals que no interfereixen entre sí.

4. Integració en temps d'execució utilitzant JavaScript

Aquesta és l'opció més flexible i en la que els equips es basen a l'hora de crear aplicacions web.

Cada MFE s'inclou en la pàgina utilitzant l'etiqueta <Script> i al carregar-se s'exposa una funció global com el seu punt d'entrada. L'aplicació que fa de contenidora determina aleshores quin MFE ha de ser muntat i crida a la funció que toca per dir-li quan i on ha de ser renderitzat.

5. Integració en temps d'execució utilitzant Web Components

Per últim, tenim una variació de l'aproximació anterior però en aquest cas treballem en el món dels web components. Aquí podem fer que cada MFE defineixi un element HTML

Figura 10: Flux d'una petició d'HTML al server amb MFE

[7] Fitxer encarregat de definir atributs funcionals per NPM, definir scripts i identifica el punt d'entrada del teu projecte i com s'encendrà

personalitzat perquè la contenidora ho instanciï en comptes de definir una funció global perquè la cridi.

4.1.7 Comentaris d'experts

4.1.7.1 Natalia Venditto[24]

Tal com apunta la xerrada de Natalia Venditto, els MFE són una arquitectura que cada cop s'està mirant més a introduir-la a equips de desenvolupament d'aplicacions web. L'estructura d'un projecte on NO incorpora aquesta arquitectura llueix de la següent manera:

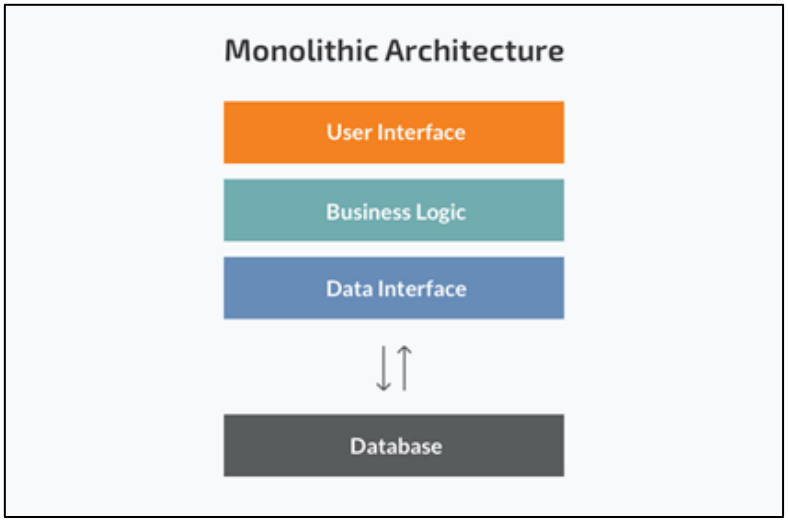


Figura 11: Arquitectura monolítica Font:n-ix.com/

Aquí és on ens hem de plantejar si aquesta estructura monolítica és necessària i imprescindible sempre. Normalment, quan es tracta d'un projecte petit si que pot ser útil, però al trobar-te amb un projecte de Front-End probablement serà molt millor l'ús dels MFE.

El gran problema que sorgeix a l'hora d'utilitzar una estructura monolítica en un projecte de grans característiques és el fet de la recompilació de tot el codi en cada fase de proves. El fet del canvi de versions juntament amb el redeplegament de tot el projecte fa que tingui una influència negativa quan parlem de costos i temps.

Per això es proposa l'estructura dels MFE on s'independitzen vàries parts d'un projecte gran i es tracten com si cada una fos una competència de negoci.

A partir d'aquí exposo diverses casuístiques que ens podem trobar dins d'un projecte d'empresa i quan es ÚTIL l'ús d'aquesta arquitectura independent:

- Primer, en una gràfica de comparació entre encapsulació[8] i els riscos del projecte, ens trobem que si és un projecte poc encapsulat i amb pocs riscos, NO voldrem fer ús dels MFE.

[8] Nivell d'ocultació, el grau d'accés que tens des de fora a alguna cosa

- Quan comparem un projecte molt monolític i amb cicles de release[9] molt freqüents, NO volem usar MFE
- Quan existeix una alta probabilitat de que hi hagi una framework de treball predeterminat i no existeixi una guia d'estils CSS, tampoc volem fer l'ús.
- Quan el tamany del projecte és petit i els mòduls els tenim centralitzats, tampoc volem fer us de MFE.

Així doncs d'aquesta xerrada extraiem que s'ha de ser curós i has de saber quin és el projecte ideal que reuneix les característiques per fer un d'una arquitectura MFE.

4.1.7.2 Manfred Steyer[25]

En aquesta presentació Manfred ens aporta uns quants casos d'ús on pensa que pot ser útil l'ús de MFE.

El primer cas d'estudi que proposa és un hospital on aquest està dotat d'un sistema d'informació que regula tot el tràfic intern de dades. Si pensem en un hospital ens podem trobar en vàries seccions com secretaria, anàlisi, radiografies, registres de pacients...

En totes aquestes divisions hi ha moltes estructures de dades i processos complicats, per això triem cada una d'aquestes divisions per convertir-les en un sistema MFE.

Un altre cas d'ús que Manfred proposa és eBanking. En un sistema de banc electrònic hi ha moltes caselles diferents i moltes comunicacions entre diferents departaments, diferents seccions... al final del dia la informació arriba de diferents dominis i diferents especialistes per això es pensa que és una bona pràctica per acabar utilitzant MFE.

Per una altra banda, Manfred exposa pros i contras d'aquesta arquitectura:

- Raons per utilitzar MFE:

✓ Quan et trobes amb un producte molt gran vols aplicar la política de "Divide and Conquer"[10] per tenir el producte subdividit i que sigui tot molt més escalable i maniobrable.
✓ Una altra raó és escalar equips i escalar a través dels diferents dominis de treball
✓ Ens aporta dinamisme amb l'evolució de tecnologies, podem canviar còmodament amb diferents entorns.
✓ Podem canviar ràpidament de requeriments d'un MFE que és molt més fàcil que canviar una aplicació monolítica.

- Conseqüències (negatives) dels MFE:

✓ Requereix un gran esforç per conèixer els frameworks que gestionen tots els MFE en una pàgina final. Noteu que en realitat són varies aplicacions independents i de diferents equips però de cara a l'usuari s'ha de veure com una sola.

[9] Sortida al mercat d'alguna cosa
[10] Divideix i venceràs, metodología per solventar problemas eficientment.

- ✓ És difícil compartir codi amb el fi de reduir tamany de mòduls. Ja que tenir 10 MFE no vol dir que nosaltres vulguem carregar Angular 10 vegades, sinó que ho volem fer només una i poder compartir aquesta instància.
- ✓ Per últim volem que tots els MFE llueixin igual i això no és tan fàcil d'aconseguir.

4.1.8 Estil dels MFE

El llenguatge CSS és per defecte global, heretat i en cascada. Alguna d'aquestes característiques no estan suportades pels navegadors. En un entorn de MFE totes aquestes característiques són molt pitjors. Per exemple, si un equip té una fulla d'estils de h2{color:red] i un altre h2{color: blue} i ambdós estan en la mateixa pàgina, existirà un problema de col·lisió d'estils!

Hi ha vàris enfocaments per a solucionar el tema del CSS independent. Molts equips opten per utilitzar BEM[26] (Block, Element and Modifier) una metodologia per garantir que el CSS s'apliqui on nosaltres volem. Una altra manera de controlar això és utilitzar SASS(Syntactically Awesome Style Sheets) els quals podem niar els selectors utilitzant un espai de noms. Finalment una de les maneres més noves i modernes per controlar això és utilitzant un dels mòduls de CSS anomenat "CSS-in-JS"[27] que garanteix també que els estils s'apliquin directament en els llocs que els desenvolupadors vulguin.

Si volem un entorn basat en plataformes, Shadow DOM també ofereix aïllament d'estils.

4.1.9 Comunicació entre aplicacions

Una de les qüestions més comunes i que més volem treballar en aquest projecte és la comunicació entre aplicacions MFE. En general el més ideal seria que no es comuniquessin massa, ja que és reintroduiex el concepte de l'acoblament inadequat que intentem prescindir en una primera instància. En aquest projecte intentarem fer una optimització de la comunicació entre els serveis MFE.

Tenim algunes idees de com ens hauríem de comunicar utilitzant MFE:

- ✓ "Custom Events" o event personalitzats permeten que els serveis es comuniquin indirectament, el que és una bona manera de minimitzar l'acoblament directe.
- ✓ El model de React de passar "props" i "callbacks", des de l'aplicació que fa de contenidor cap als MFE i viceversa, és també una bona solució. (Comunicació Downwards)
- ✓ Una tercera possibilitat és comunicar-se mitjançant URL's, però això ja en parlarem més endavant.
- ✓ Finalment, podem trobar algun mètode encara no conegut per aconseguir aquesta comunicació.

Sigui quina sigui la implementació que triem a fer, el que volem és que els nostres serveis es comuniquin mitjançant l'enviament de missatges o events entre si i evitar tenir estat compartit.

4.1.10 Comunicació amb el Back-End

Una altra de les situacions que podríem pensar és: Com es connecten els microserveis amb els Back-End? Existeixen els equips especialitzats en FullStack (Back-End i Front-End) que s'encarreguen del desenvolupament de la seva aplicació des del codi visual fins el desenvolupament de l'API i el codi de la base de dades juntament amb la infraestructura.

Un patró que ha sorgit des de que la tecnologia dels micro serveis existeix es el BFF (Back-End For Front-End). Aquesta aplica que cada aplicació té un Back-End que la seva única necessitat es servir al seu Front-End.

El BFF pot ser autònom, amb la seva pròpia lògica de negoci i base de dades o pot ser un simple Routing a un altre Back-End. Si existeixen serveis descendents pot no tenir sentit que l'equip faci l'ús d'un BFF, com en el cas de que un MFE només estigui composat d'una API que per si sola es estable.

La gràcia està en que un equip construeixi un MFE i que equips externs a ells no construeixin coses per elles, sinó que facin l'ús d'un BFF pel seu propi equip.

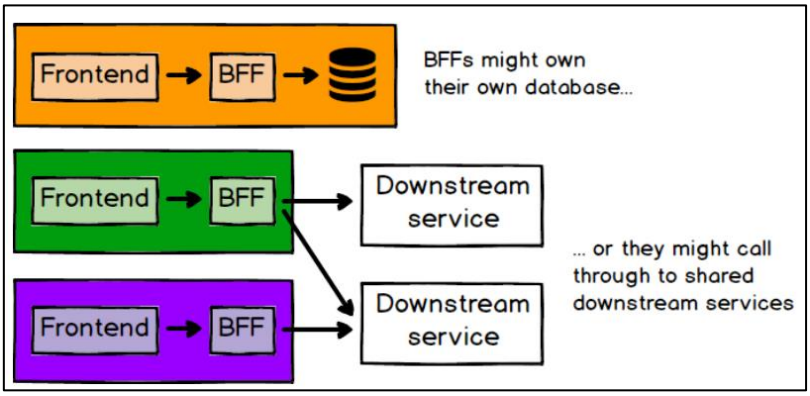


Figura 12: Organització amb BBF en l'arquitectura MFE Font: mèdium.com

Amb això finalitzem el que seria l'explicació detallada dels MFE, els micro serveis i una mica d'història de com es podria enfocar aquest tipus d'arquitectura a projectes actuals. A més a més, també tenim una idea de quines limitacions té cada mètode i com s'hauria d'implementar si vulguem treballar amb aquesta infraestructura.

4.2 Frameworks

Abans d'arrencar amb la part pràctica del projecte, cal establir quin framework integrarem al nostre portal web d'entre l'extensa varietat que hi ha al mercat. En aquest apartat definirem què és un framework i farem una descripció de 4 possibles candidats. Els frameworks escollits són: Module Federation (Webpack), Piral, Single SPA i Bit. Primerament, descriurem els trets rellevants de cadascun. A continuació, duem a terme un estudi de propietats específiques per acabar extraient conclusions que ens permetin determinar quin és el que ofereix més possibilitats l'usuari.

4.2.1 Introducció

- Què és un framework?

Un framework és l'eina, marc o entorn de treball que et facilita la programació en algun àmbit. Normalment en el desenvolupament software donant-te avantatges com evitar haver d'escriure codi de manera repetitiva, t'assegura unes bones pràctiques i una bona consistència de codi.

- Perquè volem fer ús de frameworks amb MFE?

Per una banda hem explicat que els MFE son una arquitectura que ens ofereix moltes avantatges de cara al contingut web i creació de pàgines amb Web Components. Per l'altra, els frameworks que comparem estan especialitzats en la creació de MFE i en la seva reutilització. Així doncs, ens beneficia fusionar aquests dos entorns per a la producció eficient d'elements i per treballar de manera més lleugera i dinàmica.

- Però... i què són els web components?

Els components web (CW) consisteixen en diferents tecnologies independents que permeten crear elements personalitzables i reutilitzables. Un CW pot ser utilitzat sense escriure codi, simplement afegint una sentència per importar-lo a una pàgina HTML. En poques paraules, un element que ens permet presentar informació personalitzada pròpia d'una pàgina web, una tecnologia que es desenvolupa al costat del client.

4.2.2 Module Federation (WebPack5)

Module Federation és un plugin[11] del gestor de paquets WebPack. No està del tot destinat a MFE, però si a poder tenir varies aplicacions treballant de manera independent en una única aplicació.

Però... i què és WebPack?

Webpack és un empaquetador de mòduls. Et permet generar un arxiu únic amb tots aquells mòduls que necessita la teva aplicació per funcionar. Si una aplicació necessita X arxius JavaScript, pots ficar-los tot junts en un únic arxiu on aquest per si sol ja serà suficient per a poder treballar amb l'aplicació al complet.

Entre moltes altres coses WebPack destaca:

- ✓ Poder de generar només aquells fragments JavaScript que necessita la pàgina, guanyant en optimització de càrrega d'aquesta.
- ✓ Poder automatitzar tasques repetitives.
- ✓ L'eliminació de recursos no utilitzats.
- ✓ El seu dinamisme d'aplicació a diferents frameworks destinat a web.

[11] Funcionalitat extra que s'adhereix a una aplicació de manera implícita

En una imatge podríem resumir-ho així:

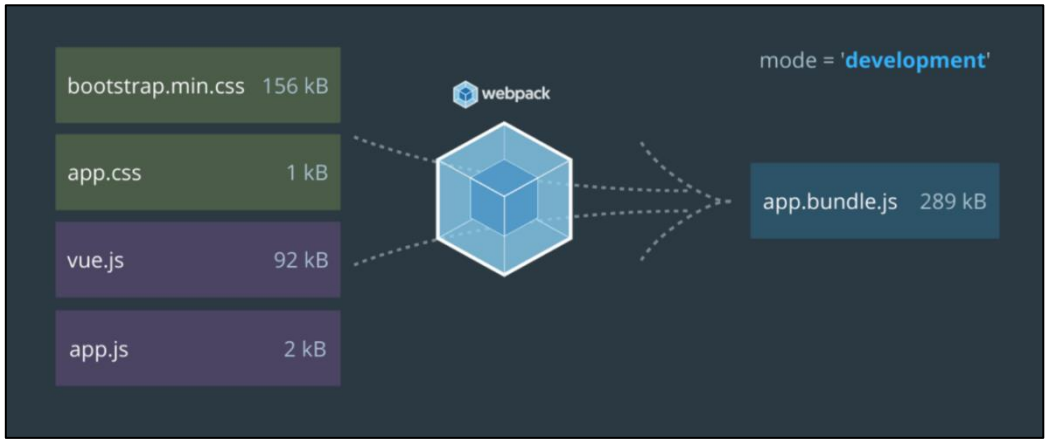


Figura 13: Funcionalitat de Webpack bàsica Font: webpack.com

Així doncs, com ja hem dit, Module Federation té l'objectiu de fer que vàries compilacions separades formin una única aplicació. Aquestes no han de tenir dependències entre si, ja que si no no poden ser desenvolupades i desplegades individualment.

A la següent imatge mostro les característiques que engloba WebPack i en negreta el plugin que comentarem en detall.

Figura 14: Característiques de Webpack Font: webpack.com

4.2.3 Piral

Piral[28] és una llibreria JavaScript per crear aplicacions modulars MFE basades inicialment en React. Des de la primera sortida ha anat adquirint noves funcionalitats i ara ha aconseguit el suport d'Angular i Vue. Els desenvolupadors d'aquests framework són Smapiot suportats amb la llicència MIT.

Una configuració senzilla de MFE implica una aplicació Root i les altres s'adhereixen a ella creant així els micro serveis.

A Piral, l'aplicació Root es diu "Piral Instance". Els mòduls (aplicacions) que s'uneixen a ella es diuen "Pilets" i tots dos poden ser generats a través de CLI de Piral.

La instància Piral gestiona tres aspectes de l'aplicació:

- ✓ El disseny a l'aplicació: Peu de pàgina, capçalera...
- ✓ Els components en comú poden que poden ser compartits entres els Pilets individuals.
- ✓ Es gestiona com es carreguen els Pilets i defineix on es podrien integrar els components compartits.

Els Pilets són aplicacions web individuals que gestionen diferents aspectes del lloc web. Encara que el CLI de Piral utilitza per defecte React, els desenvolupadors poden construir Pilets utilitzant altres llibreries com Angular, Vue i altres.

4.2.4 Single SPA

Single SPA[29] és un framework desenvolupat per Joel Denning i l'equip de Canopy. Es podria traduir com un router de javascript destinat al micro serveis Front-End. La seva primera release va ser el dia 5 d'Octubre del 2015.

Fent més èmfasis, Single SPA és una llibreria de JavaScript que et permet connectar múltiples serveis MFE independents de manera conjunta perquè es vegi i es comporti com una aplicació monolítica.

4.2.5 Bit

Bit[30] és un framework destinat al desenvolupament dels MFE. La plataforma consta de més de 200 desenvolupadors, va ser creada a desembre del 2015.

Bit és una infraestructura OSS (Operations Support System) per construir i compondre components. És una cadena d'eines extensible per aplicacions o sistemes basats en components, que són més ràpids de desenvolupar, més fàcils d'entendre, provar i mantenir, més resistents i eficaços i més fàcils de col·laborar.

En comptes de construir una aplicació que tingui molts components, BIT et permet desenvolupar components fora de qualsevol aplicació i utilitzar-los per compondre moltes aplicacions de sota amunt. Una aplicació no és més que una composició desplegada de components. Pots afegir i eliminar components de les aplicacions per ampliar o canviar la seva funcionalitat.

4.2.6 Propietats especifiques dels frameworks

L'objectiu d'aquesta secció és fer una anàlisi exhaustiu sobre propietats específiques triades per l'empresa i veure amb quines propietats destaquen aquestes tecnologies.

Ens basarem en una taula com la següent, ara buida, per valorar les propietats de cada framework.

Frameworks	Seguretat	Maduresa	Característiques	Usabilitat	Total
Module Federation					
Piral					
Single SPA					
Bit					

Taula 6: Taula resum d'exemple de tots els frameworks

Un cop fet aquesta anàlisi haurem de provar les quatre tecnologies de manera pràctica i finalment farem una taula comparativa amb una puntuació per a poder triar quina de les quatre utilitzarem per a integrar-la en el nostre entorn web.

4.2.6.1. Estudi de Module Federation

4.2.6.1.1. Seguretat

Una de les vulnerabilitats més comunes és el Cross-Site Scripting (XSS). És un defecte fonamental de la web, ja que no hi ha manera de distingir html de confiança i javascript maliciós. Però hi ha una solució per aquests casos i és la de CSP. (Content Security Policies).

La Política de Seguretat de Continguts permet a un servidor web indicar el navegador quins elements estan permesos i fer que tot l'altre tràfic estigui bloquejat per defecte. Per tant, funciona com una comprovació de la integritat de la pàgina html i evita qualsevol modificació de la pàgina per part de tercers.

Webpack i per conseqüència Module Federation, tal i com posa a la documentació oficial te un apartat de CSP. En aquest s'explica que les pòlisses de seguretat de contingut no estan activades per defecte que ho hem de fer manualment aplicant la capçalera Content-Security-Policy o un meta-tag <meta http-equiv="Content-Security-Policy" ...>.

Així doncs veiem que Webpack cobreix l'apartat de seguretat estàndard.

4.2.6.1.2. Maduresa del producte

Webpack està vigent des de el 19 de febrer de 2014. Els seus desenvolupadors principals són: Tobias Koppers, Sean Larkin i Johannes Ewald. Des de la seva primera llançada al mercat al 2014, webpack ha tingut vàries versions on s'han anat afegint varies característiques d'interès. No es fins al 7 de març de 2018 on Webpack s'estabilitza amb una versió definitiva. Aquí on es van afegint noves funcionalitats a l'aplicació com Module Federation.

Module Federation versió 5 es vigent des de el dia 10.10.2020, ja que va sortir amb la release de Webpack.

La documentació de Webpack i Module Federation es força abundant. La seva pàgina oficial té un apart exclusiu on parla de com fer anar aquest plugin. A més a més té vàris enllaços a pàgines externes com "mèdium.com" on estan explicats amb detall tutorials i documentació d'interès aportada per la comunitat.

4.2.6.1.2.3.Característiques addicionals

- Estils

Com ja sabem, quan treballem amb JavaScript i llenguatges de programació Front ens trobem comunament amb fitxers de tipus .css, .sass o scss. Webpack per si mateix només coneix JS així que si volem fer el mateix amb altres tipus de fitxers hem d'aplicar-li algun mòdul.

És aquí on parlem dels "loaders". Aquests són com una espècie de plugin basat en nodes per ajudar a webpack a compilar o transformar un recurs perquè acaba essent de tipus JS.

CSS-loader és un mòdul npm[12] que ajuda a webpack a recollir el css de tots els arxiu css referenciats a la teva aplicació i posarlo en una cadena de caràcters. Aleshores style-loader agafa aquesta cadena generada pel css-loader anterior i la posaria dins de l'etiqueta style en l'arxiu index.html.

Amb aquests mòduls ja tindríem resolt el tema dels estils utilitzant la tecnologia de webpack amb Module Federation. Cal comentar que serveix amb tota classe de plugins ja que es gestiona a través de l'arxiu de configuració global webpack.config.js.

- Compatibilitat Cross-Browser

Tal i com posen ells a la documentació, Webpack suporta tot els navegadors existents i les seves noves versions.

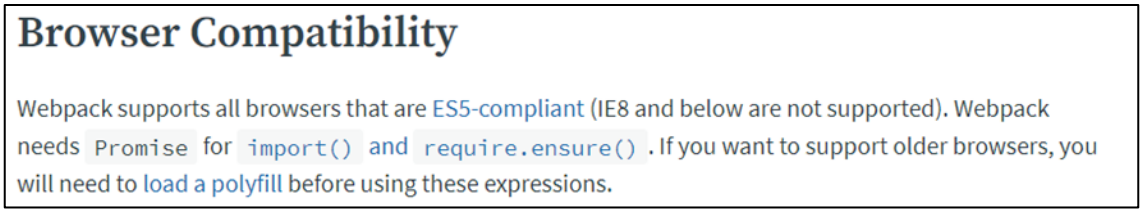


Figura 15: Compatibilitat entre navegadors a Module Federation

- Adaptabilitat i escalabilitat

Pel que he pogut comprovar, Webpack té una adaptabilitat molt alta a qualsevol entorn de desenvolupament. Encara que a la documentació oficial no fan èmfasis en aquest punt, no he trobat cap impediment a l'hora d'utilitzar Webpack o Module Federation.

[12] Sistema de gestió de paquets de Node.js

4.2.6.2. Estudi de Piral

4.2.6.2.1. Més de Piral

L'arquitectura de Piral consisteix en 3 parts:

- La part frontal de l'aplicació, que consisteix amb una aplicació Shell anomenada "Piral Instance" i els seus MFE que completen aquesta instància que s'anomenen Pilets.
- Addicionalment tenim els mòduls que componen els Feed Service, que són totes les dependències necessàries per proporcionar els MFE de manera correcta.
- Finalment per temes de desenvolupament per poder publicar les instàncies, els micro serveis, i emmagatzemar tot el tema de les comandes per accionar certes funcionalitats de Piral, tenim Piral-CLI.

A la següent imatge podem veure un esquema de l'arquitectura d'aquestes tres parts.

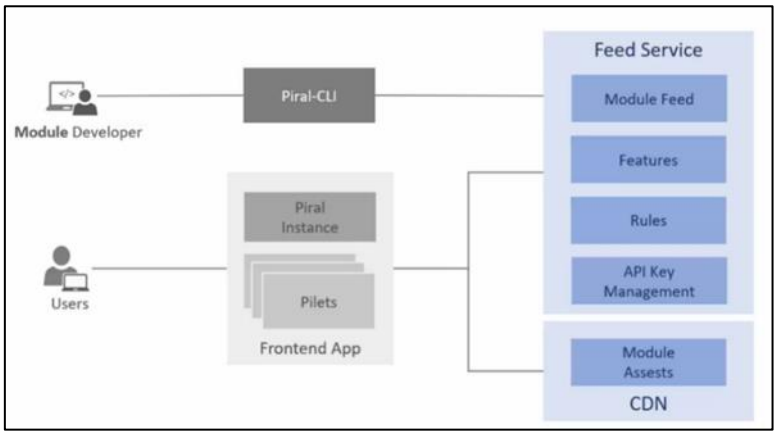


Figura 16: Arquitectura de Piral Font: Piral.io

Una de les diferències que es poden apreciar en l'àmbit d'arquitectura és la comunicació entre el FrontEnds i els seus serveis. Mentre que en una aplicació monolítica la seva arquitectura és com la que es mostra a continuació, és a dir, sense cap mena de comunicació entre els equips que gestiona cada un dels Front-End's d'una aplicació:

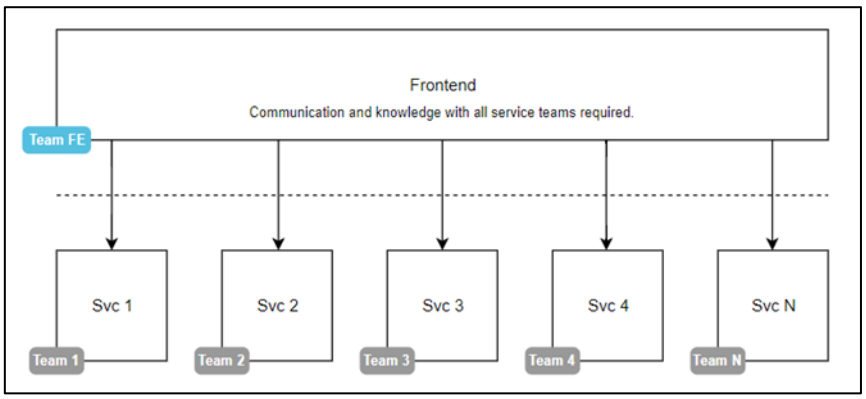


Figura 17: Equip Front-End en entorn monolític Font: piral.io

En canvi, utilitzant Piral i el concepte de micro serveis es mostraria de la següent manera, fent que els Mòduls puguin ajudar a altres equips a desenvolupar el seu servei, és a dir, permetent la comunicació entre cadascun dels “Pilets” d’una “Instancia Piral”.

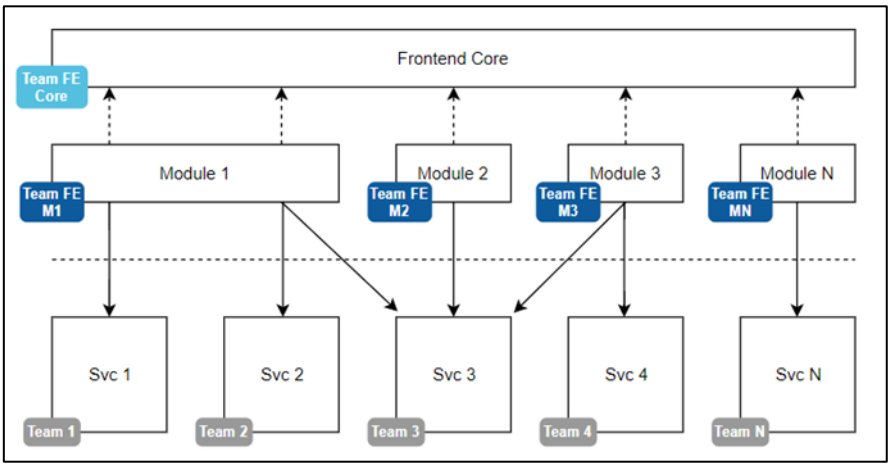


Figura 18: Equip Front-End en entorn MFE Font: piral.io

En l’àmbit de construcció del framework, Piral no comença des de zero. En la següent imatge es mostra la configuració inicial a nivell de blocs de la que es compon quan tu crees una Piral Instance. Com podem veure esta basada en React i quan nosaltres ens referim a l’aplicació creada per Piral, ens referim als blocs: Piral, Piral Core i Piral EXT.

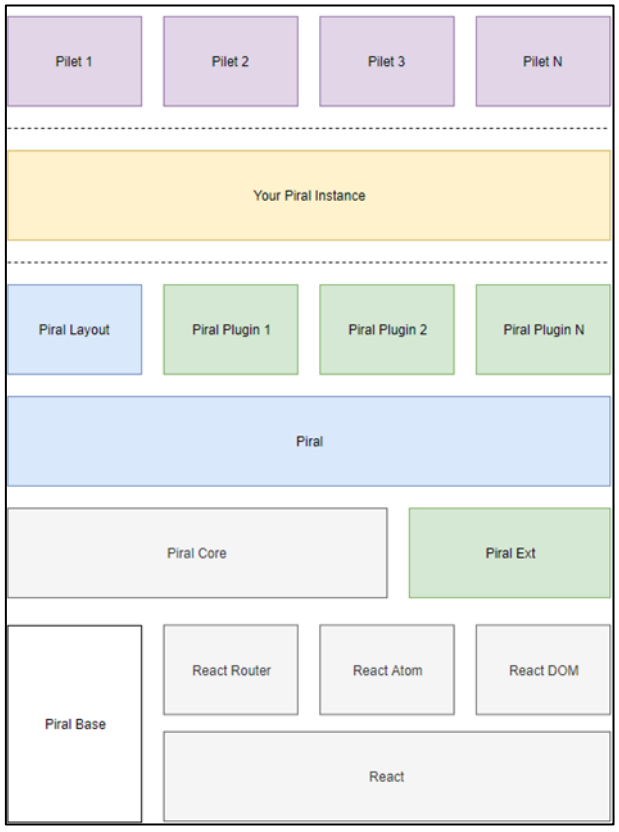


Figura 19: Piral en l'àmbit de blocks des de una base Font: piral.io

Un Pilet és simplement un paquet de NPM que conté una llibreria. La llibreria és consumida per Piral mentre que el paquet és inspeccionat i desempaquetat per un servei. El paquet conté metadata, un o més fitxers JS i potencialment alguns assets. A la següent imatge es mostra un esquemàtic de què podria contenir un Pilet quasevol.

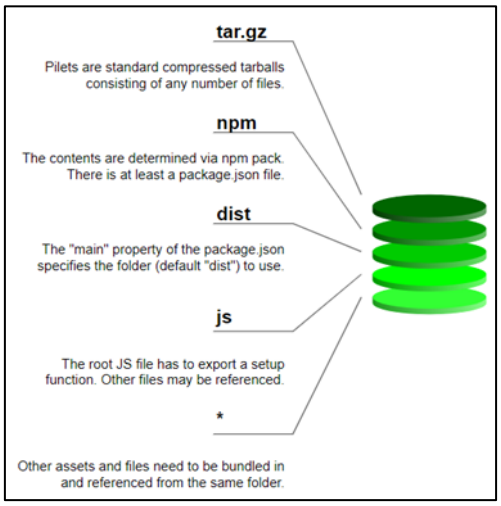


Figura 20: Contingut d'un Pilet Font: piral.io

En l'àmbit com funciona la càrrega interna de la construcció del framework (inici d'una instancia de Piral) ens trobem que és un procés multi-nivell tal i com mostro a la següent imatge:

Finalment, l'API d'un Pilet és l'encarregat de tenir la comunicació amb la Shell de Piral per poder preparar els seus components de manera correcta.

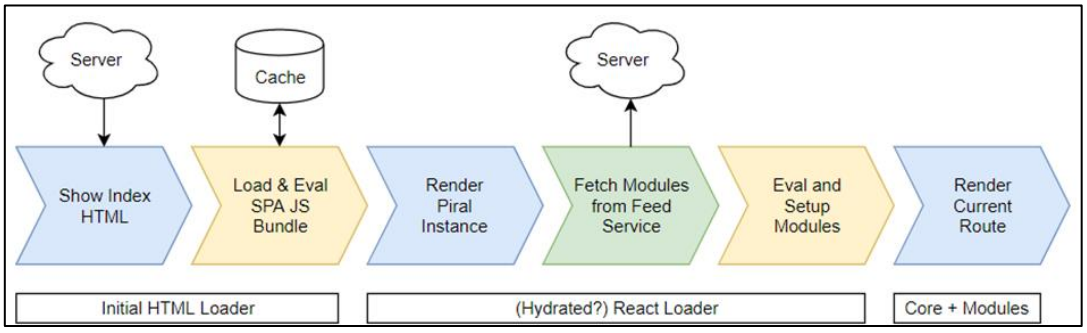


Figura 21: Càrrega interna de Piral Font: piral.io

4.2.6.2.2. Seguretat

Pel que fa la a seguretat d'aquest framework ens trobem que a la pàgina principal i tal com he comentat a la introducció de Piral: Està sota la llicència de MIT fent que cap software sigui capaç de perjudicar en la seguretat de Piral. Apart d'això no trobem cap mena de suport addicional com ens ofereixen alguns dels altres frameworks estudiats.

4.2.6.2.3. Maduresa del producte

Piral va ser llançat el 21 de febrer de 2019. És un framework relativament nou i aparentment sembla que ho porta un desenvolupador anomenat Florian Rappl. A més a més treballa darrere l'empresa de Smapiot. Es troba suportat per la llicència MIT.

Per ser un framework encara en proves i ser relativament nou té una documentació molt extensa i molt ben explicada. A més a més té un xat a Gitter on es reuneix tota la comunitat preguntant per dubtes, bugs i altres característiques relacionades amb el framework.

Un punt negatiu de Piral és que els tutorials de Youtube estan desactualitzats i moltes coses de les quals expliquen allà ja no funcionen.

4.2.6.2.4. Característiques addicionals

- Estils

En aquest apartat ens trobem que piral no ofereix cap mena de suport especial de cara als estils. Com que cada instància de piral te Pilets independents, cadascun d'ells va associat al seu corresponent fitxer d'estils. De fet, a la documentació oficial hi ha un apartat que parla exclusivament d'això citant textualment: "El codi i els estils estan aïllats a fi i afecte de prevenir conflictes o sobreescriure en els arxius"

- Capacitat Cross-Browser

La pagina oficial te suport en aquest punt i mostra les següents possibilitats.

Compatibility Table

The following browsers are supported by `piral`.

Browser	Version	Notes
Internet Explorer	≥ 11	requires polyfills
Edge	*all*	req. pilet schema v1
Chrome	≥ 52	-
Firefox	≥ 47	-
Safari	≥ 6.1	req. pilet schema v1
Opera	≥ 41	-
Mobile Safari	≥ 7.0	req. pilet schema v1
Android Browser	> 4.4	requires polyfills

Figura 22:Compatibilitat Cross-Browser de Piral Font: piral.io

- Adaptabilitat i escalabilitat

Pel que he pogut provar, Piral està molt més enfocat per usuaris de Linux. Al tenir un Windows em vaig trobar amb vàris erros d'adaptabilitat del framework. A part d'això

s'ha de comentar que la intenció d'aquest framework es poder escalar solucions de portals web així que aquest aspecte està cobert amb aquest framework.

4.2.6.3. Estudi de Single-SPA

4.2.6.3.1. Més de Single-SPA

Aquí exposem punts a favor de l'ús d'aquest framework:

- Multiples Frameworks operant alhora amb un únic "entry point"

Single-SPA et permet barrejar i combinar llibreries i fer que actuïn com una aplicació construïda de manera monolítica. Per exemple, pots utilitzar React i Angular per fer dues seccions de la mateixa app i semblarà que només estigui construïda amb un d'ells.

- Rendiment controlat

Encara que això depengui més o menys de les mateixes aplicacions, si utilitzem Single-SPA per unir-les, podem reduir la mida del paquet de qualsevol aplicació individual i seguir connectant-les com es desitgi sense perdre cap mena de rendiment de les aplicacions.

- De molt fàcil ús i amb Lazy Load per defecte

La gran majoria de la preparació del framework és executar la comanda: create-single-spa. Tota la resta són configuracions específiques que dependran del tipus d'aplicació compartida que vulguem implementar.

Una aplicació dins del món de Single-SPA, és un grup de components de UI (User Interface) registrades en aquest framework que controlen part de la pàgina web. Aquestes aplicacions coexisteixen sense importar el tipus de framework en el que estan programades.

- Però... com funciona tot això a la pràctica?

Un cop la teva aplicació és registrada a Single-SPA, la clau és una API comuna que està feta de 3 parts:

- ✓ Una funció "bootstrap" per la inicialització de l'aplicació.
- ✓ Una funció de tipus mount que activa l'aplicació.
- ✓ I una funció de tipus unmount que desactiva l'aplicació.

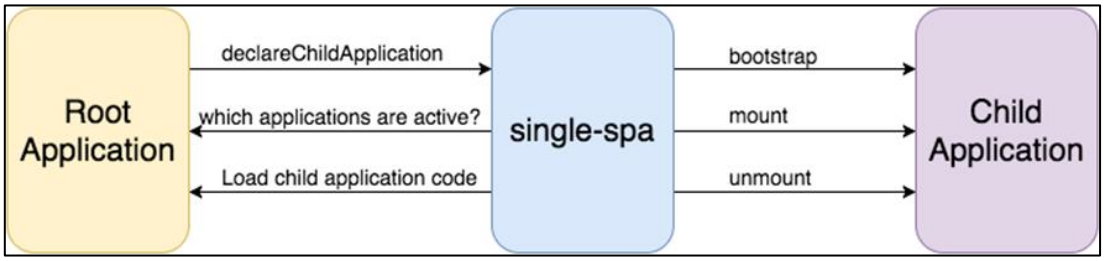


Figura 23: Mecanisme intern de Single-SPA Font:single-spa.js.org/

El teu codi seguirà aquestes 3 part i la resta s'encarregarà el framework Single-SPA. La llibreria de Single-SPA ajuntarà les teves aplicacions actives i les mostrarà en temps d'execució quan tu li manis mostrar-les.

4.2.6.3.2. Seguretat

Sobre la seguretat, la mateixa pàgina de documentació de Single-SPA respon a la pregunta de si "És necessari tenir seguretat addicional que la mateixa instal·lació del Framework?", la resposta bàsicament és que Single-SPA t'ofereix la mateixa seguretat que com si no l'utilitzessis, és a dir, la mateixa seguretat que tinguin els teus projectes i les teves aplicacions.

Addicionalment, parla de punts que poden ser interessants a tenir en compte pel que fa a el tema de la seguretat:

- ✓ ES6 Module Dynamic Imports
- ✓ Cross-Origin Resource Sharing
- ✓ Content Securicy Policies
- ✓ Subresource Integrity
- ✓ Import Maps també és una cosa a tenir en compte amb els CSP.

4.2.6.3.3. Maduresa del producte

Single SPA va ser llançat el 18 de juny de 2018 com la primera release. A hores d'ara segueix sent un framework molt nou, en fase de proves i en correcció de bugs. Durant les noves releases han anat incorporant les diferents funcionalitat que té a hores d'ara el framework. Les mes importants han estat la compatibilitat de poder utilitzar MFEs amb React, Angular i Vue independentment.

Té un xat d'Slack per poder compartir qualsevol tipus de dubte, problema o bug que trobis i els desenvolupadors poden donar-te suport.

La documentació és correcta, però hi ha diverses guies i aparts de la documentació que no estan actualitzades o existeixen problemes no esmentats en aquesta.

4.2.6.3.4. Característiques addicionals

- Estils

En una arquitectura de micro serveis és important tenir tant el CSS global compartit com el CSS específic per cada micro serveis. Només ha d'haver-hi una còpia de tot el css global i el css específic de cada MFE ha de tenir un ús únic per a cada micro i intentar que no col·lideixin amb cada un.

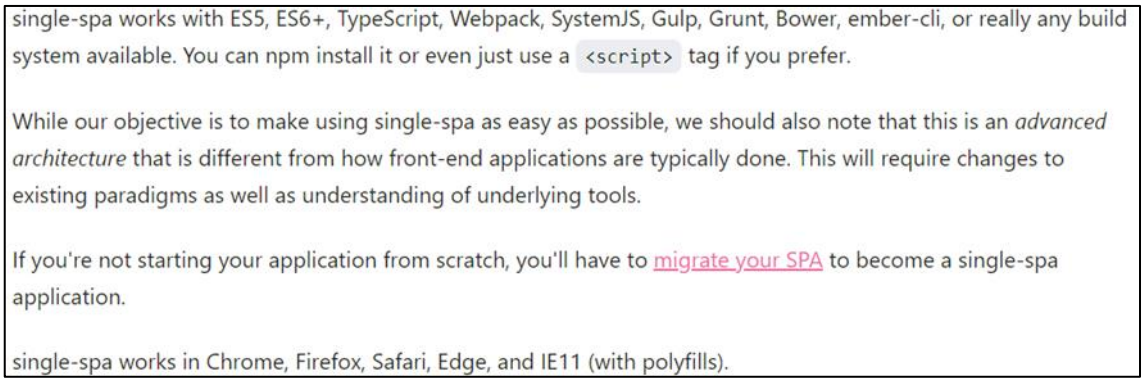
Tal com posen a la seva pròpia documentació de tots els recursos que tenen per gestionar els estils d'una pàgina utilitzant Singe-SPA.

Té suport amb temes de "Desing Systems", Css Globals, tractament d'estils amb tota menade frameworks com: React, Angular, Vue, Svelte... Css amb Lazy Loading, funcions de mount i unmount amb Css i molt de suport mes.

A més a més ens trobem amb un mòdul creat pel propi framework anomenat single-spa-css on aquest implementa funcions d'ajuda amb el tema de la càrrega, el mount i l'ummount del propi css. Addicionament ens trobem amb el suport de Webpack amb un plugin on ens aporten tractament de css amb tota mena d'assets. Aquest últim és anomenat "mini-css-extract-pluguin" i "ExportRunTimeCssAssetPlugin".

- Capacitat Cross-Browser

Tal com posa a un apartat de la documentació oficial, ens trobem en que Single-SPA respon al següent


single-spa works with ES5, ES6+, TypeScript, Webpack, SystemJS, Gulp, Grunt, Bower, ember-cli, or really any build system available. You can npm install it or even just use a `<script>` tag if you prefer.

While our objective is to make using single-spa as easy as possible, we should also note that this is an *advanced architecture* that is different from how front-end applications are typically done. This will require changes to existing paradigms as well as understanding of underlying tools.

If you're not starting your application from scratch, you'll have to migrate your SPA to become a single-spa application.

single-spa works in Chrome, Firefox, Safari, Edge, and IE11 (with polyfills).

Figura 24: Capacitat Cross-Browser a Single-SPA Font: /single-spa.js.org/

- Adaptabilitat i escalabilitat

Pel que fa a l'adaptabilitat i l'escalabilitat ho resumirem amb la funcionalitat real del framework. Aquest resol un problema difícil de interoperabilitat entre MFE amb molta soltesa. Però també amaga molts problemes quan s'efectua una acció no suportada pel framework. Sobre l'escalabilitat dels errors no està molt ben aconseguida ja que quan es comet un no se sap exactament perquè és.

4.2.6.4. Estudi de Bit

4.2.6.4.1. Més de Bit

- Espai de treball

Construeix i compon components. L'espai de treball és la fundació Bit. És on es desenvolupen i es componen els components. Et permet construir projectes totalment distribuïts amb una experiència de desenvolupament simple i monolítica. Només cal anar a la interfície d'usuari de l'espai de treball per començar a desenvolupar components.

- Àmbit

Gestiona i escala components. Els àmbits són el lloc on puges, versions i organitzes els teus components. És com un magatzem de components. Els àmbits remots et permeten utilitzar components en tots els projectes. Pots configurar i allotjar àmbits remots a qualsevol servidor.

Bit.dev és una plataforma opcional de grau empresarial per allotjar i connectar tots els àmbits i components per donar als equips una experiència de col·laboració entre projectes relacionats.

- Desenvolupament impulsat per components

Els equips distribuïts poden construir components i compondre'ls en moltes aplicacions, produint experiències d'usuari consistents, eliminants duplicacions de codi i millorant la velocitat de desenvolupament.

Amb el desenvolupament dirigit per components es pot agafar el mateix enfocament de composició de MFE pel temps d'execució del sistema i traslladar-lo a un estil d'arquitectura de cofi amb més granularitat.

En comptes de construir peces mes grans(serveis, Front-End), mira de compondre components més petits i reutilitzables. Pensa en els components com serveis en la base de codi i utilitza'ls per ompondre una aplicació concreta.

- No és exclusiu

El desenvolupament basat en components ha fet molt per escalar les interfícies d'usuari. Frameworks com React, Vue i Angular el milloren fent les nostres aplicacions més modulars. Bit agafa el mateix enfocament i afegeix aquesta funcionalitat a qualsevol cosa que es construeixi. Qualsevol funcionalitat pot ser consolidada en uns components i reutilitzada en moltes aplicacions per molts equips.

- Eina de construcció

Bit és una eina de construcció. No hi ha SDK's o dependències de temps d'execució afegides al teu codi. A més a més de no afectar el rendiment de la teva aplicació, això també significa que pots decidir no utilitzar Bit sense el cost de re factorització del teu codi.

- Distribuir el monòlit

El desenvolupament web modern ha fet de primera classe els components web. Encara hi ha molts projectes que s'organitzen amb una estructura monolítica. Les bases de codi monolítiques poden ser farragoses de mantenir, especialment amb la creixent complexitat i tamany de l'equip. Els equips hauran de col·laborar per a fer aquestes bases de codi de manera distribuïda.

- Distribuir el desenvolupament de components

Amb Bit grans aplicacions web modernes poden estar compostes per diferents equips que construeixen components. Cada equip pot desenvolupar, exposar e integrar components de forma autònoma. Tots els components poden ser fàcilment integrats en temps de construcció.

- Extensibilitat

Bit se centra a crear una experiència senzilla i flexible pels desenvolupadors.

4.2.6.4.2. Seguretat

Bit, en ser una comunitat tan extensa, tenen força controlat el tema de la seguretat en el seu framework. Tenen un document oficial on es descriuen pràctiques i polítiques que Bit adopta per assegurar-se de que s'està utilitzant un sofware estable i segur i que totes les dades allotjades estan protegides.

A més a més compta amb un certificat SOC2 que fa de testimoni com un dels més alts estàndards dels sistemes d'informació relatius a la seguretat, disponibilitat, integritat de processament, la confidencialitat i la privacitat.

Addicionalment, hi ha més característiques com:

- ✓ Els empleats que treballen directament sobre l'emmagatzematge d'arxius nomes tenen accés als arxius comprimits i de cap manera tenen accés al format de text pla.
- ✓ Cap tercer te accés a cap codi allotjat de forma privada.
- ✓ Instal·lacions de sistemes que utilitzen sistemes operatius reforçats
- ✓ Tallafocs dedicats i VPN per bloquejar accés no autoritzat al sistema
- ✓ Manteniment de totes les dependències de software actualitzades a les últimes actualitzacions de seguretat.
- ✓ Manteniment de la seguretat utilitzant: Pràctiques de registre, limitacions en l'accés al software i auditories de seguretat i proves de penetració.

5.2.6.4.3. Maduresa del producte

Bit va ser llançat com la primera versió estable publica al 6 de febrer de 2017. Consta amb un equip de més de 200 desenvolupadors.

Té dos seccions clarament diferenciades: Bit.dev és la que s'encarrega de ser el cloud dels web components que es generen amb Bit i actual com un Github. Pots guardar els teus MFE en aquesta secció per poder recuperar-los més tard, guardar-los o simplement per mostrar-los a altres usuaris. A més a més conta d'una secció on pots visualitzar el component amb el codi a temps real!

Per una altra, banda tens Harmony-Bit.dev, en aquesta secció tens tota la documentació necessària per treballar amb Bit. A més és la secció adequada per a la creació, modificació i eliminació dels MFE. La documentació és bona i està molt ben organitzada, se'ns dubte es una dels frameworks més treballats dels que hem vist fins ara juntament amb Webpack Module Federation.

Compta amb llicencia i és 100% open Source.

4.2.6.4.4. Característiques addicionals

- Estils

Pel que fa a els estils, i la gestió d'aquests tenim el mateix que sempre. Típicament una aplicació conté arxius d'estils que es comparteixen entre els diferents components de l'aplicació. Els arxius d'estils poden ser CSS pur o utilitzar un preprocessador com scss o less.

A més a més també pot existir un conjunt de variables utilitzades com tokens de disseny per denotar elements reutilitzables com colors o breakpoints.

Aquestes variables s'utilitzen en vàris components pel que s'han de crear com components propis. Si els estils es reparteixen entre varis components, es recomana agrupar-los en un espai de noms dedicat.

La finalitat de tot això és un tema de processament d'arxius d'estil. Pot ser crític que els arxius d'estils estiguin destinats a ser processats pel projecte que els conte. És a dir, que s'executi un procés de concordança que al·liïna els etils amb l'HTML. Per tant, els components que només contenen estils no necessiten tenir compilador associat a ells.

Per eliminar-ho s'ha d'anunciar al Package.json de la següent manera:

Figura 25: Tractament d'estils a Bit Font: single-spa.js.org/

Els components de només estils ara estan disponibles pel seu consum.

- Capacitat Cross-Browser

En aquest cas, bit no conte massa informació sobre la compatibilitat de navegadors, comenta molt per sobre en un apartat de compilació que està suportat en múltiples navegadors, i si tens algun problema amb antics navegadors canviïs el versionat del javascript.

- Adaptabilitat i escalabilitat

Bàsicament adaptabilitat i escalabilitat son definicions pròpies d'aquest framework. Està precisament dissenyat per escalar aplicacions web i poder fer components que s'adaptin a qualsevol entorn d'un portal web.

4.2.7 PoC (Proof of Concept)

Una vegada completat l'anàlisi, donarem pas a fer la prova de concepte o Prove of concept de cada tecnologia (recollides als Annexos I, II, III, IV). Consisteix a fer una valoració en primera persona amb el fi de decidir quin és el més útil per a la realització de la integració.

Per a fer-ho crearem components a l'atzar però ens centrarem més en l'explicació de com funciona cada tecnologia, quin seguiment de codi té i que significant cadascuna de les coses que veurem.

4.2.8 Valoració final dels possibles frameworks

Aquesta seria la valoració personal de les tecnologies basant-me en, per cada un dels camps, en diferents criteris.

Taula 7: Valoració final dels tots els frameworks

Frameworks	Seguretat	Maduresa	Característiques	Usabilitat	Total
Module Federation	7/10	9/10	7/10	9/10	32/40
Piral	5/10	7/10	8/10	6/10	26/40
Single SPA	8/10	7/10	8/10	8/10	31/40
Bit	7/10	7/10	7/10	9/10	30/40

Framework que, en un primer moment, hem començat a utilitzar ja que ha aconseguit la puntuació més alta.

Framework que hem acabat utilitzant per fer la implementació final.

Un resum sobre les valoracions personals seria el següent:

Pel que fa a nivell de **seguretat**: S'han avaluat quines eines anticiberatac disposa, a quines vulnerabilitats s'exposa i quina és la seva gestió de dades privades de la tecnologia.

Pel que fa a nivell de **maduresa**: S'ha avaluat la trajectòria de la documentació i exemples que disposen les fonts oficials, la comunitat de desenvolupadors que hi participa, el suport del fabricant, i quines llicències i costos té.

Pel que fa a nivell de **característiques**: S'ha avaluat de quins estils disposa, quina es seva escalabilitat i adaptabilitat amb altres tecnologies i la seva capacitat amb els altres navegadors.

I finalment, pel que fa a la **usabilitat**: és la meva nota personal de quan els he provat per fer la prova de concepte.

4.3 Mètodes de comunicació entre MFEs

En aquest apartat explicarem com podem compartir informació entre MFE i perquè ho hem de fer. A més a més mostrarem algunes formes de comunicació entre múltiples aplicacions.

Partint de la base que ja sabem que és un MFE, hem de ser conscients que les empreses utilitzen aquesta infraestructura per a desacoblar les aplicacions de manera independent, fer més eficient i facilitar l'ús d'aquestes.

Seguint una lògica apropiada, una bona arquitectura és aquella en la que els MFE estan desacoblats i no necessiten comunicar-se amb freqüència, però hi ha algunes coses que els MFE podrien compartir o comunicar, com podrien ser: funcions, components o alguna lògica d'estat.

- Compartició de Codi

Per funcions, per components i lògiques comuns, el tema del compartiment de codi es podria exportar en una llibreria i importar-se a cada aplicació que volguéssim utilitzar el codi.

I pel tema de crear un paquet (llibreria) hi ha vàries maneres de fer-ho, però ho profunditzarem més endavant.

- Compartició d'Estat

Però... que passa amb l'estat compartit? Perquè algú necessitaria compartir l'estat entre vàries aplicacions?

Per fer-ho fàcil, posem un exemple real d'e-commerce:

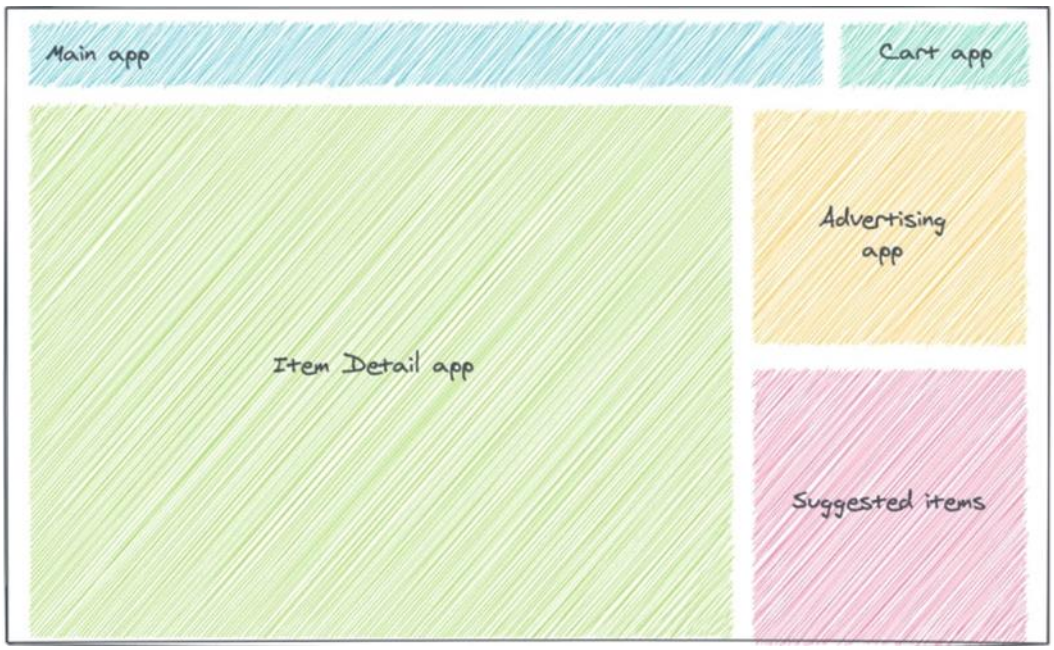


Figura 26: Cas real per aplicar comunicació Font: dev.to

Aquesta imatge representa una pàgina web amb la seva aplicació principal, la seva cistella, els detalls de l'article, la publicitat i els ítems suggerits.

Cada quadrat representa un MFE amb un domini o una funcionalitat específica i podria estar utilitzant qualsevol framework de Front-End.

Si afegim una mica d'informació a la plantilla de la pàgina web mostrada anteriorment ens n'adonem que és possible que necessitem que varis MFE es comuniquin entre si:

Figura 27: Possibles comunicacions entre MFE Font: dev.to

1- En primer lloc, podríem voler que tant l'apartat d'"Inspeccionar producte/article" com "Articles recomanats" (Part verda i rosa respectivament) tinguin comunicació amb la cistella de la compra (part blava a la dreta).
2- Els articles recomanats podrien utilitzar els articles actuals que hi ha a la cistella per suggerir mes productes relacionats amb els que hi ha actualment fent ús d'alguna mena d'algoritme complex.
3- En l'apartat d'inspeccionar element, podria sortir algun missatge si aquell element ja està a la cistella.

Com a recomanació, si dos MFE s'han de passar informació molt freqüentment, s'ha de considerar la possibilitat de fusionar-los. Els MFE perden força quan no són mòduls independents, sinó que depenen d'altres. Aleshores, que passa si els articles suggerits es podrien fusionar amb els d'inspeccionar, però els necessitem de manera independent?

Per aquests casos parlarem de 5 implementacions diferents:

1. Web Workers
2. Props and callbacks
3. Custom Events

4. Pub Sub Library (Windowed-observable)
5. Custom Implementation

Una taula comparativa amb informació sobre aquests modes seria aquesta:

- ✅ 1st-class, built-in, and simple
- 💛 Good but could be better
- 🔶 Tricky and easy to mess up
- 🛑 Complex and difficult

Criteria	Web workers	Props and callbacks	Custom Events	windowed-observable	Custom implementation
Setup	🛑	✅	✅	✅	🔶
Api	🔶	💛	💛	✅	🔶
Framework Agnostic	✅	✅	✅	✅	🔶
Customizable	✅	✅	✅	✅	🔶

Figura 28: Taula amb informació sobre aquests mètodes Font: dev.to

4.3.1 Web Workers

Els web workers[31] permeten executar una operació d'script en un thread en segon pla separat del thread que executa l'aplicació principal d'una aplicació web. L'avantatge d'això és que el processament complex es pot executar en un thread per separat permetent que el principal (generalment interfície d'usuari) es pugui executar sense ser bloquejat o ralentitzat.

Es pot crear un exemple per il·lustrar una comunicació simple entre 2 MFE amb un web worker molt simple utilitzant workerize-loader i create-micro-react-app, dues eines de creació de web workers i MFE en react, respectivament.

L'exemple conté 2 MFE, 1 aplicació contenidora i una llibreria compartida exposant el worker.

Definició gràfica de Web Worker

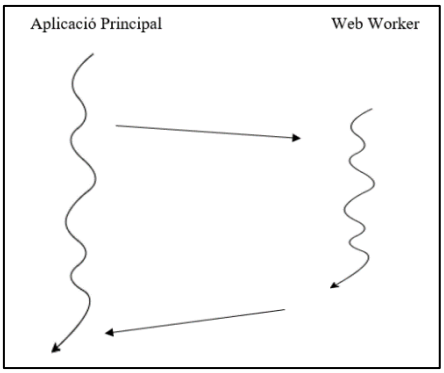


Figura 29: Fluxes d'un Web Worker Font: Elaboració pròpia

Definició gràfica de Web Worker amb MFE i traspàs d'informació:

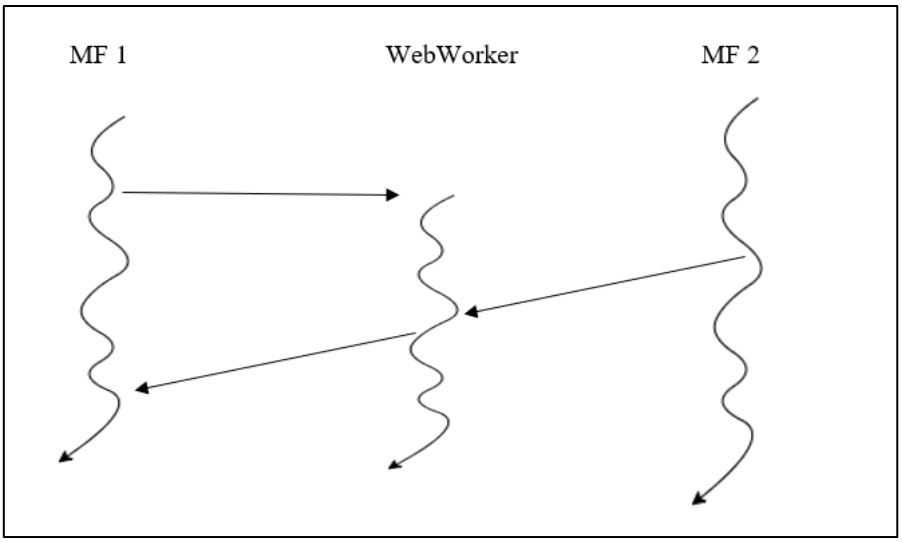


Figura 30: Traspàs d'informació entre MFEs i Webpack Font: Elaboració pròpia

El MF1 s'encarrega de la creació del webworker i de la implementació de mètodes i està a la "espera" del que passi al web worker. El MF2 és un formulari on pot enviar text pla normal que va parar al Worker i MF1 recull aquest text pla per mostrar-lo al seu HTML.

Avantatges:

- Els 2 MFE no es comuniquen entre ells sinó que utilitzen un "bypass" per intercanviar informació
- Des de el punt de vista de MND, paral·lelitzes accions alhora que evites que el Thread principal s'aturi o es ralentitzi

Inconvenients:

- Preparació de la infraestructura del workers complexa
- Massa text per una API
- Complicat de compartir el mateix worker entre varis MFE sense utilitzar una "window"

4.3.2 Custom Events

Un Event personalitzat[32] pot ser alguna cosa que el navegador fa o alguna cosa que l'usuari pot fer.

Alguns exemples són:

- Una pàgina web ha acabat de carregar
- Un camp d'input ha canviat
- Un botó ha estat clicat

Quan utilitzem JS en una pàgina HTML, JS pot reaccionar a aquest events.

En el cas dels custom Events, bàsicament són Events personalitzats. La nomenclatura que s'utilitza per referenciar-los és la següent:

```
const MessageEvent = new CustomEvent("message", options);
```

Figura 31: Construcció d'un custom event Font: Elaboració pròpia

Podem crear un exemple utilitzant un contenidor d'aplicació i 2 MFE utilitzant custom events.

La primera aplicació constaria d'un formulari que quan un usuari fa click al botó d'enviament de dades, es transmet un esdeveniment personalitzat que envia aquestes dades a l'altre aplicació on aquesta ultima l'està esperant amb un EventListener.

Avantatges:

- Fàcilment construïble.
- Personalitzable
- Agnòstic de frameworks
- Els MFE no necessiten saber els seus pares.

Inconvenients:

- Molt farragós en l'api d'events

4.3.3 Props and Callbacks

Els props[33] són un patró per compartir informació entre un component pare i un component fill, on el component pare tria atributs i els envia al component fill com un objecte. És important saber que això treballa de manera unilateral, és a dir, que la informació només es comparteix des del pare cap al fill i no al revés.

De fet, per poder enviar informació de fills a pares, s'utilitzen les callbacks!

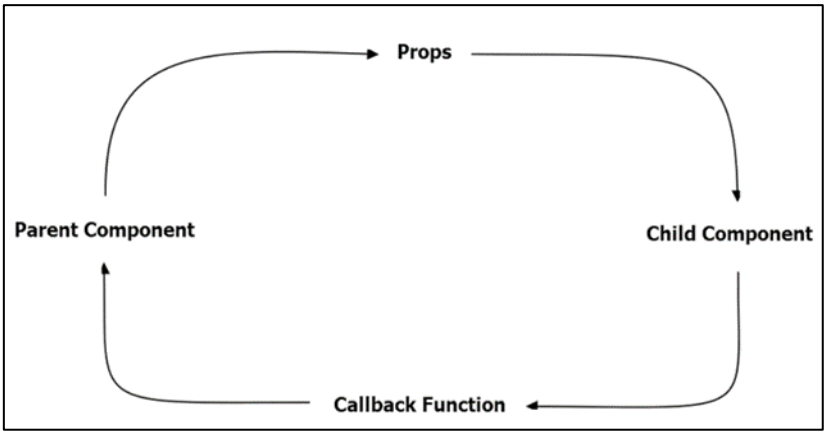


Figura 32: Fluxe entre Props i Callbacks Font: Elaboració pròpia

Podem crear un exemple per il·lustrar això utilitzant *crma* (create react micro app) per preparar MFE de React.

Utilitzem una aplicació que actua com a contenidor i dos MFE.

La idea és que el contenidor actuï de intermediari entre els 2 MFE utiltizants props i callbacks. És exactament com l'exemple anterior dels workers amb el mateix exemple dels missatges.

Avantatges:

- API Simple
- Construcció de la infraestructura simple
- Personalitzable

Inconvenients:

- Difícil de treballar quan hi ha varis frameworks en joc
- Quan una propietat canvia, tot els MFE han de ser renderitzats

4.3.4 Windowed Observable

En una arquitectura de MFE, un dels principals problemes és la comunicació entre les aplicacions, això afecta directament als MFE. Windowed observable[34] és una llibreria que pretén resoldre aquest problema aportant una API simple i agnòstica sense cap mena de configuració addicional.

La solució consisteix en exposar un "observable" que es comporta com un sistema de publish/subscriber que emet esdeveniments per espais de noms.

En aquesta nova era de micro serveis, aplicacions i Front-End's hi ha una cosa en comú: els sistemes distribuïts.

Mirant l'entorn dels micro serveis un mètode de comunicació bastant popular son les cues de publish/susbcribers tanmateix com els serveis AWS[13], SQS[14] i SNS[15].

Ja que cada MFE i el contenidor estan a la mateixa "pantalla", podem utilitzar una finestra per mantenir una comunicació global utilitzant una implementació de pub/sub

El sistema de publish/ subscriber tracta d'un sistema de missatgeria en el que els emisors del missatge, anomenats publishers no programen els missatges perquè s'enviïn a receptors específics, anomenats subscribers, sinó que classifiquen els missatges publicats en classes sense saber que hi ha a l'altra banda.

Avantatges:

[13] Amazon Web Services: Plataforma de computació al núvol
[14] Amazon Simple Queue: Servei de cues de missatge distribuït d'Amazon
[15] Amazon Simple Notificacion Service: Infraestructura pe'entrega masiva de missatges

- Api simple
- Preparació simple
- Molt personalitzable
- Aïllament de espais de noms
- Característiques extres del framework
- Open source

Inconvenients:

- Aquesta llibreria depèn de la finestra, per tant tothom pot canviar aquesta implementació

4.3.5 Implementació personalitzada

Després de veure totes aquestes implementacions es poden barrejar entre si i crear la teva pròpia creació de comunicació entre els MFE. Però probablement pot ser enganyós i complicat de tractar amb implementacions inventades.

5 Marc pràctic

Després de l'anàlisi general dels MFE, l'avaluació dels frameworks per a la creació d'una arquitectura MFE i l'anàlisi dels sistemes de comunicació, procedim a la part més pràctica del projecte. Aquesta part, que es descriu a continuació, ha consistit amb les proves dels sistemes de comunicació estudiades, la creació d'un sistema amb una arquitectura MFE amb el framework més adient i finalment la integració de les diferents eines de Front-End en aquest sistema. En definitiva, hem volgut optimitzar tot el presentat en el TFM en l'àmbit pràctic.

5.1 Proves amb els sistemes de comunicació

Aquest punt el dedicarem a provar dos mètodes de comunicació ja explicats en la secció 5.3 del TFM. Per a fer-ho mirarem d'implementar els dos mètodes en dues tipologies diferents dels frameworks estudiats: Module Federation (Instancies Actives) vs Piral (Cloud compartit).

La primera prova de concepte s'ha fet amb Module Federation utilitzant el concepte de Web Workers. Per a fer-ho hem creat 2 MFE independents, una llista i un formulari. La idea base és que el formulari pugui enviar text pla a la llista i que aquesta renderitzi els missatges en temps real.

5.1.1 Web Worker amb Module Federation

Per fer aquesta prova, hem volgut generar una arquitectura de sistema de fitxers com la del Annex I. S'han creat dues aplicacions, una conté la llista i l'altre conté el formulari. En l'aplicació que "hereta" l'altre també hi hem posat el web worker fent que aquest pugui operar independentment quan les dues aplicacions estan connectades. Basem-nos en la següent estructura:

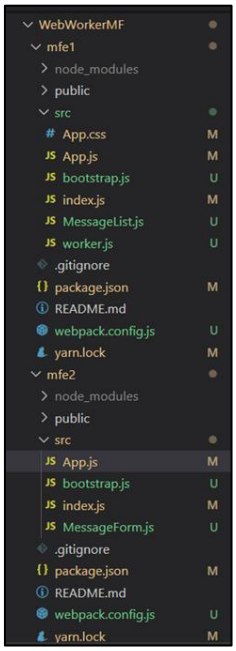


Figura 33: Sistema de fitxers utilitzant un web Worker Font: Elaboració pròpia

Com podem intuir, mfe1 és l'aplicació que, mitjançant Webpack "importa" mfe2 alhora que exposa el worker perquè mfe2 hi pugui accedir.

Un cop aconseguim tenir ambdues aplicacions a la mateixa pàgina, ens comuniquem utilitzant la llibreria general "window" per poder completar la comunicació amb el Web Worker. El codi del worker llueix de la següent manera.

Figura 34: Lògica d'un web Worker Font: Elaboració pròpia

Aquesta és la funció que utilitzen els altres MFE per a poder traspassar informació. En aquest cas és un exemple molt senzill però, en un cas normal aquest Worker s'encarregaria de diverses gestions de traspàs d'informació de manera independent a l'aplicació base.

En l'àmbit teòric tindríem un esquema de les següents característiques.

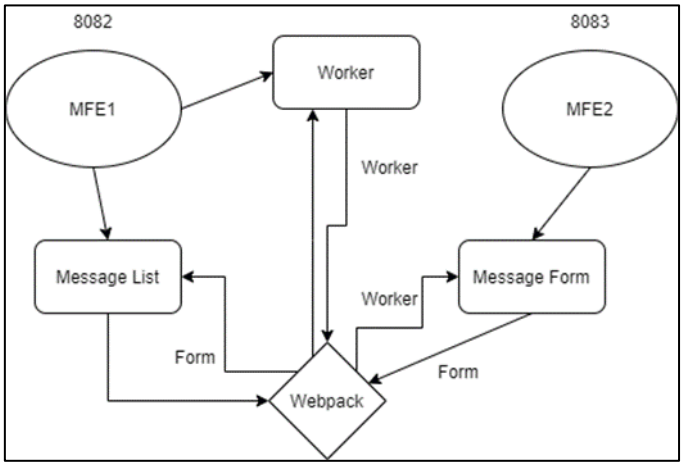


Figura 35: Diagrama de flux de l'arquitectura MFE amb Web Worker pròpia Font: Elaboració pròpia

Una vegada construïda aquesta arquitectura, obtenim el resultat final següent.

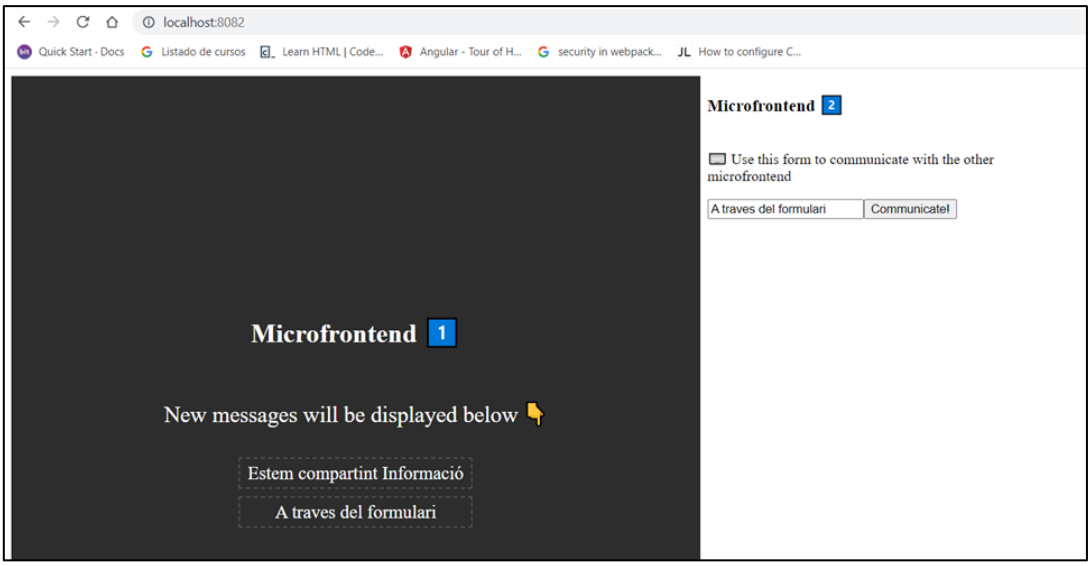


Figura 36: Comunicació entre dos MFE utilitzant Web Worker Font: Elaboració pròpia

5.1.2 Windowed Observable amb Module Federation

Per a fer aquesta prova també hem seguit l'arquitectura anterior de l'annex I. Aquesta vegada hem volgut provar un framework anomenat Windowed-Observable. Aquest framework et permet "crear" cues de publicació i subscripció permetent així la comunicació amb 2 MFE dins un mateix porta web. A continuació mostrem l'arquitectura de sistema de fitxers.

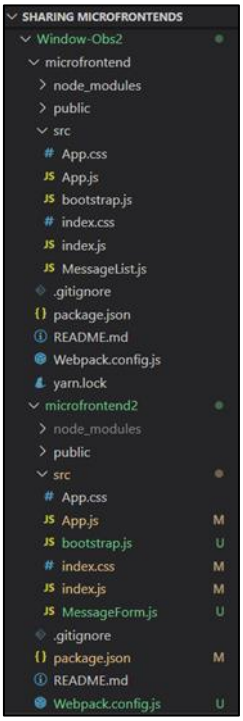


Figura 37: Sistema de fitxer amb Windowed Observable Font: Elaboració pròpia

Aquí podem veure que el fitxer que feia de web worker ja no hi és. Això és perquè ara la funció que feia la fa de manera interna windowed-observable. El que si hem de fer és

instal·lar el framework de manera global a tot el projecte i fer les comunicacions pertinents:

npm install windowed-observable

En el cas del formulari ho fem així.

```
import { Observable } from 'windowed-observable';

const observable = new Observable('messages');

function MessageForm() {
  const handleSubmit = (e) => {
    e.preventDefault();
    const { target: form } = e;
    const input = form?.elements?.something;

    observable.publish(input.value)
    form.reset();
  }
```

Figura 38: Programació del formulari de la prova de concepte amb Windowed-Observable Font: Elaboració pròpia

El que estem fent aquí és declarar un observable amb l'etiqueta *messages*. Ara qualsevol mena d'informació que publiquem (amb la funció *publish*) s'emmagatzemarà amb la referenciació de l'etiqueta *messages* mentre estiguis dins del mateix entorn web.

Per altra banda, per a poder fer la recepció del missatge que toca ho fem de la següent manera:

```
const observable = new Observable('messages');

function MessageList() {
  const [messages, setMessages] = useState([]);

  const handleNewMessage = (message) => {
    setMessages((currentMessages) => currentMessages.concat(message));
  };

  useEffect(() => {
    observable.subscribe(handleNewMessage);

    return () => {
      observable.unsubscribe(handleNewMessage);
    }
  }, [handleNewMessage]);
```

Figura 39: Programació de la lògica de la llista que rebrà els missatges Font: Elaboració pròpia

Aquí el que estem fent és declarar un "estat" de la variable missatges, bàsicament això el que farà és quan el DOM s'actualitzi (Enviem un missatge en el formulari) agafi aquesta informació (Aquí és on entra la funció del Windowed Observable) i vagi concatenant-la en un mateix vector per finalment poder-lo tractar i mostrar el valor de l'últim missatge.

El resultat d'això és idèntic al anterior.

5.1.3 Web Worker amb Piral

Per a la construcció d'aquesta prova, hem inicialitzat el projecte com expliquem a l'annex II. Com ja hem comentat amb anterioritat Piral conte una API pròpia. De fet, per a fer aquestes proves no caldria l'ús de frameworks externs o Web Workers, ja que la mateixa API de Piral conté funcions d'intercanvi d'informació entre Pilets, però requereix tenir molt bon domini del framework i coneixement de tot el seu propi sistema intern.

Nosaltres en aquesta prova hem anat al més bàsic de tot: crear els dos Pilets amb la llista i el formulari i utilitzar un fitxer que estableixi la comunicació amb ambdós fent ús de la llibreria genèrica de "Window".

A grans trets el que s'ha hagut de fer és alguna cosa com aquesta.

```
export function setup(app: PiletApi) {
  const value = app.setData('MF1Data', MF1Page);
  app.registerPage('/MF1Page', MF1Page);
  app.registerMenu('/LinkToMF1', LinkToMF1)
}
```

Figura 40: Registre i connexió entre aplicacions de Piral Font: Elaboració pròpia

La possibilitat d'intercanvi d'informació a Piral va per ordre de creació de cada Pilet. És a dir, el Pilet 1 podrà seleccionar una Data a traspassar al Pilet 2, però no al revés. Fins on sabem, el mètode intern d'intercanvi d'informació és unidireccional. Per tant, al Pilet 2 farem alguna cosa així.

```
export function setup(app: PiletApi) {
  app.registerMenu('/LinkToMF2', LinkToMF2);
  app.registerExtension("foo", app.getData("MF1Data"))
  app.registerPage('/MF2Page',MF2Page);
}
```

Figura 41: Recollida de dades al Pilet i registre d'una extensió Font: Elaboració pròpia

Basicament, *app.getData* s'emporta la informació assignada al Pilet 1 amb *setData*. El que li estem passant és la llista sencera perquè la pugui renderitzar al Pilet2.

La part especial de Piral és: Com podem passar una “pàgina” sencera a un Pilet independent?

La manera de fer-ho fàcil és amb Extensions. Aquestes són un recurs de Pilet per afegir informació i objectes addicionals a una pàgina que ja està creada.

Per a fer-ho, cridem la funció *registerExternsion* a través de l’API de Pilet i li donem una etiqueta que la identifiqui. En aquest punt, podem utilitzar aquesta extensió en qualsevol punt de l’aplicació. En el cas de voler usar-la dins d’una funció que renderitza un HTML hem de passar l’API com a paràmetre i indicar-ho de la següent manera.

```
export const MF2Page: React.FC<PageComponentProps> = ({piral}) => {
```

Figura 42: Pas de paràmetres a traves de l'API de Piral Font: Elaboració pròpia

I referenciar l’extensió així:

```
22      <piral.Extension name="foo"></piral.Extension>
23      <h3>Microfrontend 2 </h3>
```

Figura 43: Referenciació d'una extensió a l'HTML de Piral Font: Elaboració pròpia

A partir d’aquí ja podem treballar amb normalitat utilitzant el WebWorker. El resultat final sense aplicar estils es el següent.

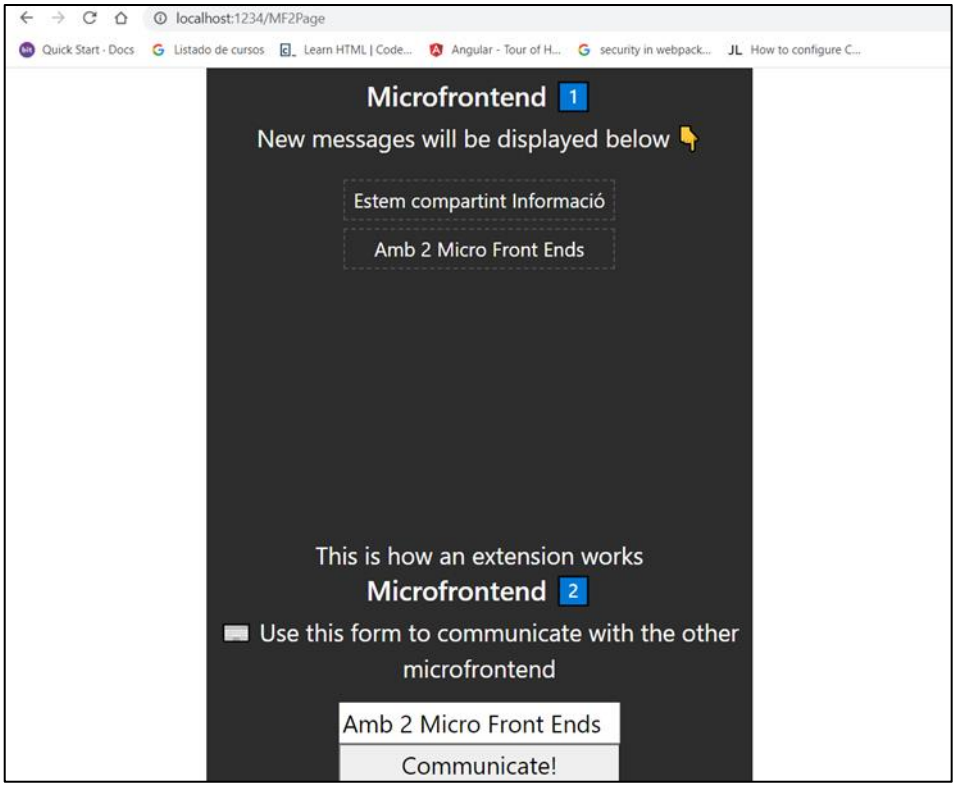


Figura 44: Resultat final de la connexió amb Web Workers a Piral Font: Elaboració pròpia

5.1.4 Windowed Observable amb Piral

Per aquesta prova hem muntat el sistema amb el framework de Windowed-Observable però, hem trobat una incompatibilitat que no hem sabut resoldre.

Entenem que Piral i WO no són compatibles a causa de la personalització del propi framework. L'error mostrat és aquest:

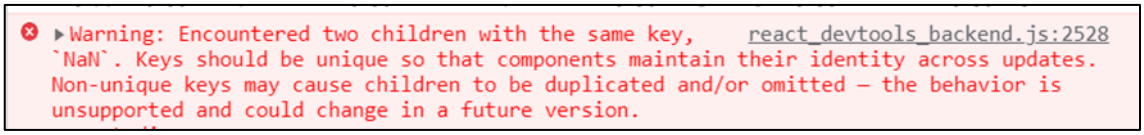


Figura 45: Error mostrat al integrar Windowed-Observable a Piral Font: Elaboració pròpia

5.2 Optimització de serveis MFE

5.2.1 Preparació de l'aplicació contenidora

En aquesta secció explicarem com fer la preparació de la integració del micro serveis a la pràctica. Per fer-ho posarem d'exemple el següent cas d'ús: Es vol implementar una aplicació que formi un sistema de registre de dades de l'usuari, un registre del vehicle en la seva possessió i finalment una taula resum amb tota la informació que l'usuari hagi triat en els 2 formularis per enviar la informació.

- Primer de tot, per què ens interessa treballar amb MFE?

Com ja sabem, els MFE's són petites aplicacions que contenent una funcionalitat especifica i mostren un contingut en una pàgina web. Un MFE pot estar produït amb qualsevol tecnologia de front (JS, Svelte, React, Angular, Vue...). En un projecte convencional només una d'aquestes tecnologies pot ser seleccionada per fer feina en un projecte.

- Però i si aconseguíssim que un projecte pogués funcionar amb diverses tecnologies diferents alhora?

D'això tracta aquest apartat, volem obtenir la màxima eficiència possible en el món del front, fent que cada equip pugui fer feina amb la tecnologia que més còmoda li sigui i que, finalment, puguin compartir els seus projectes i que cooperin tots en un.

- Objectius de l'escenari proposat

Per a aquest escenari vam voler buscar uns objectius que satisfessin tots els punts tractats en aquest projecte i que a l'hora fos d'utilitzat, els exposem a continuació:

- ✓ Una aplicació contenidora de varis MFEs.
- ✓ Com a mínim 2 frameworks diferents en una mateixa pantalla
- ✓ Un back què és comuniques com a mínim amb un dels frameworks
- ✓ Que tots els components es puguin muntar i desmuntar en una mateixa pantalla
- ✓ Comunicació amb els diferents frameworks amb els mètodes ja estudiats. Aquest últim serà Windowed-Obserbable.

Per a fer-ho, s'ha proposat un esquema on puguem veure fins on poden arribar els MFE's a dia d'avui i veure l'aplicació pràctica real que tenen. A més a més, ens podem trobar algunes limitacions que són d'interès a l'hora de plantejar un projecte amb aquesta tecnologia i arquitectura.

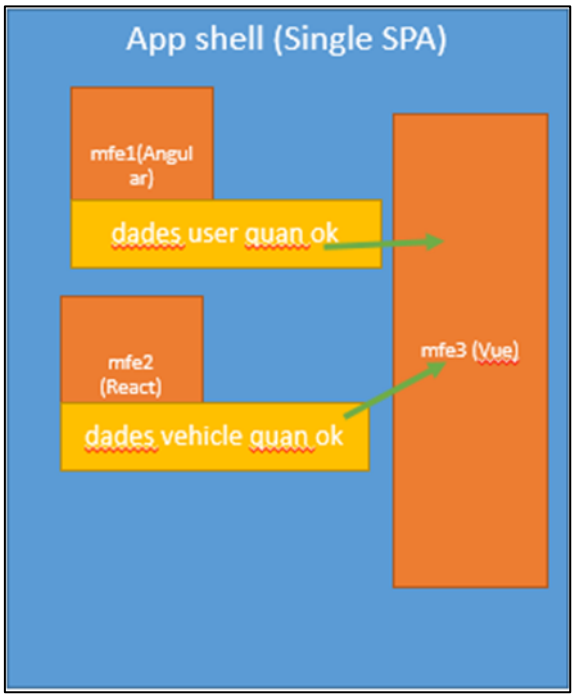


Figura 46: Esquema visual de l'arquitectura proposada Font: Elaboració pròpia

Com veiem l'arquitectura consta d'una aplicació que fa de contenidora, 3 MFE amb 3 frameworks diferents i un servidor on es connectarà algun dels MFE per obtenir informació. A més a més 2 MFE compartiran informació mitjançant tècniques de front explicades en aquest projecte.

A continuació fem una guia detallada dels passos que s'han de executar per aconseguir aquesta arquitectura funcional.

- Elecció del framework per a l'arquitectura

En vista dels resultats de Module Federation explicat a la secció 6.3 del TFM, hem d'escollir una altra eina per a fer aquesta arquitectura. Partint de la base que Piral, un altre dels frameworks estudiants és un "ecosistema propi" i Bit és una eina de compartició de MFE i no ens aporta cap eina per a la coexistència dels MFE en un mateix entorn, ens decantem per Single-SPA. Aquest últim ens ofereix ajudes tant per la creació d'una instància contenidora, com per qualsevol mena de framework registrat a la plataforma.

A continuació expliquem com crear l'entorn per començar a treballar, la remodelació dels components perquè tot tingui sentit, la incorporació del Back-End, la inclusió dels components en una mateixa pàgina i finalment les configuracions addicionals perquè tots els components coexisteixin en un sol projecte.

5.2.2 Single-SPA com aplicació contenidora

En primer lloc, creem un directori, que en el nostre cas s'anomenarà "poc" i entrem la següent comanda per terminal:

create-single-spa –layout

Això ens crea l'entorn que necessitem per, a posterior, poder gestionar altres aplicacions que utilitzem.

En la fase de creació ens demanarà si l'aplicació és contenidora o actua com a MFE i el més important, un nom d'organització, aquesta serà la referència durant tota la fase de creació de projecte: "tfmpoc"

En concret ens crea una estructura com aquesta:

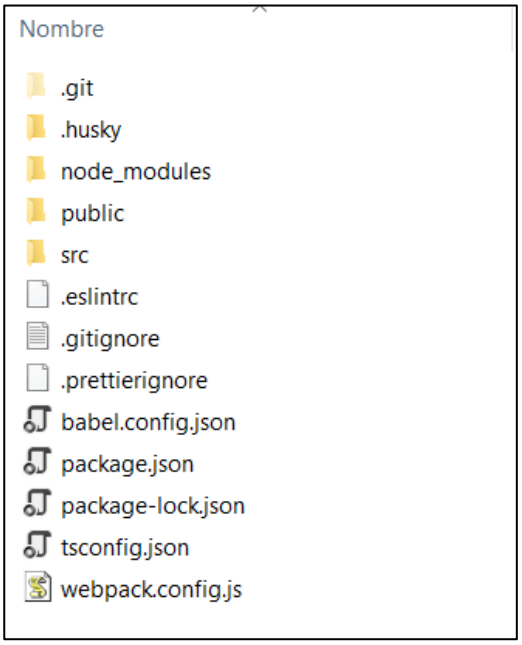


Figura 47: Root de Single-SPA

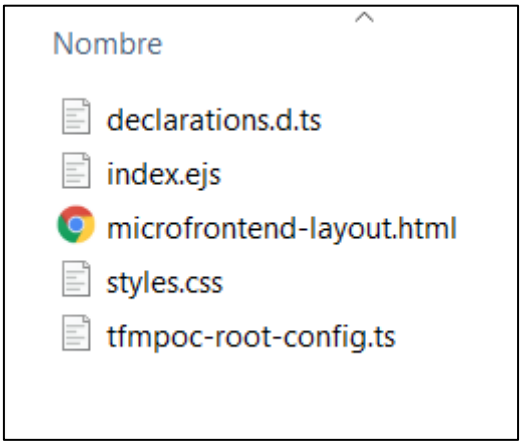


Figura 48: Carpeta src de Single-SPA

En aquesta aplicació ens hem de centrar en com importem les aplicacions que s'han d'afegir. Per a fer-ho, hem d'anar a la carpeta src i veure els arxius *index.ejs*, *microfrontend-layout.html* i *tfmpoc-root-config.ts*.

Index.ejs → Conté tot el necessari per a fer els imports dels projectes, ja ve predefinit, ho fem la següent manera:

```
<% if (isLocal) { %>
<script type="systemjs-importmap">
  {
    "imports": {
      "@tfmpoc/root-config": "//localhost:9000/tfmpoc-root-config.js",
      "@tfmpoc/poc-react": "//localhost:8080/tfmpoc-poc-react.js",
      "@tfmpoc/poc-vue": "//localhost:8081/js/app.js",
      "@tfmpoc/poc-angular2": "//localhost:4200/main.js",
      "@tfmpoc/poc-angular":"//localhost:4201/main.js"
    }
  }
</script>
<% } %>
```

Figura 49: Index.ejs resultant al final de l'integració (Aplicacions) Font: Elaboració pròpia

En primer lloc carreguem les dependències que fan falta per a tots els frameworks que vulguem utilitzar:

```
<script type="systemjs-importmap">
  {
    "imports": {
      "single-spa": "https://cdn.jsdelivr.net/npm/single-spa@5.9.0/lib/system/single-spa.min.js",
      "react": "https://cdn.jsdelivr.net/npm/react@16.13.1/umd/react.production.min.js",
      "react-dom": "https://cdn.jsdelivr.net/npm/react-dom@16.13.1/umd/react-dom.production.min.js",
      "vue-router": "https://cdn.jsdelivr.net/npm/vue-router@3.1.6/dist/vue-router.min.js"
    }
  }
</script>
```

Figura 50: Index.ejs resultat al final de l'integració (Dependències) Font: Elaboració pròpia

Finalment per a fer el renderitzat dels components ho podem fer de dues maneres:

A l'arxiu *microfrontend-layout.html* utilitzant l'etiqueta utilitzant el nom que li hem donat als imports dels MFE a l'arxiu *index.ejs* que us he mostrat just abans.

```
<application name="@tfmpoc/poc-angular"></application>
```

Figura 51: Càrrega del MFE com a aplicació incrustat a l'HTML Font: Elaboració pròpia

O bé podem fer-ho a l'arxiu tfmpoc-root-config.js registrant les aplicacions una a una. Per a fer-ho necessitem tractar aquest registre com un objecte i li hem de donar un nom, una aplicació i una ruta de quan ha d'estar actiu:

```
registerApplication({
  name:"@tfmpoc/poc-react",
   app: () => System.import("@tfmpoc/poc-react"),
  activeWhen: ['/']
})
```

Figura 52: Càrrega del MFE tractat com objecte Font: Elaboració pròpia

Nosaltres en aquesta prova ho hem tractat a nivell d'aplicació utilitzant la primera opció. Així podem aplicar CSS com a un DOM normal. A partir d'aquí, cada cop que haguem d'importar una nova aplicació hem de repetir els mateixos passos canviant l'atribut *name.*

A continuació mostrem com fer aquesta integració a l'aplicació base ja creada.

5.2.3 React

- Preparació de React sense suport de Single-SPA

Primer de tot necessitem Node.js, l'entorn en temps d'execució per a poder executar tota mena d'aplicació Front, en concret, tota mena d'aplicació que utilitzi JavaScript.

Figura 53: Descarrega de Node.js Font: Node.js

En el nostre cas ens descarregarem l'última versió estable (Long Term Suport). Amb aquesta serà suficient per a fer totes les proves que realitzarem en aquesta fase final pràctica.

Creem un projecte de React amb la següent comanda:

npx create-react-app my-app

Automàticament, NPM ens instal·larà React:

```
C:\Everis\MicroFrontEnds\Component>npx create-react-app my-app
Need to install the following packages:
  create-react-app
Ok to proceed? (y) y
npm WARN deprecated tar@2.2.2: This version of tar is no longer supported, and will not receive security updates. Ple
 upgrade asap.

Creating a new React app in C:\Everis\MicroFrontEnds\Component\my-app.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

[                    ] / idealTree:babel-loader: timing idealTree:node_modules/babel-loader Completed in 82ms
```

Figura 54: Instalació de React mostrada per terminal Font: Elaboració pròpia

El projecte resultant d'aquesta instal·lació te la següent pinta:

Nombre	Fecha de modificación	Tipo	Tamaño
.git	21/12/2021 21:39	Carpeta de archivos	
node_modules	21/12/2021 21:39	Carpeta de archivos	
public	21/12/2021 21:39	Carpeta de archivos	
src	21/12/2021 21:39	Carpeta de archivos	
.gitignore	21/12/2021 21:38	Documento de tex...	1 KB
package.json	21/12/2021 21:39	JSON File	1 KB
package-lock.json	21/12/2021 21:39	JSON File	1,094 KB
README.md	21/12/2021 21:38	Archivo MD	4 KB

Figura 55: Sistema de fitxer d'un projecte de React Font: Elaboració pròpia

Els fitxers/carpetes que haurem de tenir en compte en aquesta prova són els següents:

• *Node_modules* → Carpeta on van totes les dependències instal·lades a partir de NPM.

• *Src* → Components i lògica de tota la nostra aplicació. És on nosaltres crearem els components i implementarem la lògica de React.

• *Package.json* → És el cor del projecte. Guarda metadata important, defineix atributs funcionals per NPM, defineix scripts i identifica l'entry point del teu projecte

A públic trobarem *l'index.html*, el punt d'entrada que renderitzarà el nostre navegador. Tanmateix com a Angular, tenim una etiqueta que referència el nostre component base: App. Aquest el podem trobar dins de la carpeta src on la mostrem en la següent imatge. A més a més, en aquesta carpeta se sol posar tota mena de multimèdia.

Nombre	Fecha de modificación	Tipo	Tamaño
App.css	21/12/2021 21:38	Archivo CSS	1 KB
App.js	21/12/2021 21:38	Archivo JavaScript	1 KB
App.test.js	21/12/2021 21:38	Archivo JavaScript	1 KB
index.css	21/12/2021 21:38	Archivo CSS	1 KB
index.js	21/12/2021 21:38	Archivo JavaScript	1 KB
logo.svg	21/12/2021 21:38	Microsoft Edge HT...	3 KB
reportWebVitals.js	21/12/2021 21:38	Archivo JavaScript	1 KB
setupTests.js	21/12/2021 21:38	Archivo JavaScript	1 KB

Figura 56: Carpeta src de React Font: Elaboració pròpia

En aquesta carpeta haurem de fer alguns canvis perquè tot tingui sentit a l'hora de muntar una arquitectura de MFE's.

Com ja vam explicar en l'annex I, perquè el front-end's es puguin carregar eficientment hem de canviar el workflow de React:

1. Per a fer-ho canviarem el fitxer index.js i li direm bootstrap.js.
2. Crearem un altre index.js on només importarem el fitxer bootstrap.js

Amb aquest canvi en queda quelcom així:

Nombre	Fecha de modificación	Tipo	Tamaño
App.css	10/11/2021 18:55	Archivo CSS	1 KB
App.js	10/11/2021 15:27	Archivo JavaScript	1 KB
App.test.js	03/06/2021 01:13	Archivo JavaScript	1 KB
bootstrap.js	10/11/2021 15:26	Archivo JavaScript	1 KB
cardetails-form.js	10/11/2021 18:45	Archivo JavaScript	3 KB
index.css	03/06/2021 01:13	Archivo CSS	1 KB
index.js	09/11/2021 17:20	Archivo JavaScript	1 KB
logo.svg	03/06/2021 01:13	Microsoft Edge HT...	3 KB
reportWebVitals.js	03/06/2021 01:13	Archivo JavaScript	1 KB
setupTests.js	03/06/2021 01:13	Archivo JavaScript	1 KB

Figura 57: Sistema de fitxers després dels canvis adients Font: Elaboració pròpia

Per clarificar una mica el sistema, procedim a explicar el workflow de React.

1- *Index.js* → Es el punt d'entrada al sistema, en el nostre cas importa bootstrap.js

2- *Bootstrap.js* →Fa el render d'un element al DOM, aquest "l'elegim" del index.js de la carpeta públic. Aquest element serà App.js
3- *App.js* → Renderitza tots els components de l'aplicació React, en aquest cas, només "cardetails-forms.js"
4- *Index.css i app.css* → CSS global i CSS particular del component, respectivament.
5- *Cardetails-form.js* → El nostre component com a tal, a continuació veiem que hem aplicat perquè ens faciliti la programació

- Creació del component amb React

Per a fer una prova de concepte més real, hem volgut utilitzar propietats exclusives de React per veure si realment en un entorn multiframework es compatible o no.

En aquest cas hem utilitzat una llibreria pròpia de React anomenada React-Hook-Form. Per a poder-a utilitzar només hem d'instal·lar-la:

npm install react-hook-form

I fer el pertinent import dins del fitxer on vulguem utilitzar-ho. Com podem veure a la següent imatge, conté propietats específiques de la llibreria:

```
return (

 <div id="container" >
   <h3 id="h3">Micro Front-End A</h3>
   <img id="img"src={logo} alt="Logo"/>
   <form onSubmit={handleSubmit(onSubmit)} id="form">
       <label><b>Tipus d'indentificació</b></label>
   <select  name="idtype" className="form-select"
    {...register("idtype", {required:{ value: true, message:"Camp necessari"}})}>
       <option value=""></option>
       <option>DNI</option>
       <option>NIE</option>
       <option>NIF</option>
   </select>
     <span className="text-danger text-small d-block mb-2">
         {errors?.idtype?.message}
     </span>

   <label><b>Numero d'identificació</b></label>
   <input name="id" className="form-control" id="select2"
   {...register("id", {required:{value:true, message:"Camp necessari"},pattern: {
    value:  /^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{8,}$/,
```

```
function HookForm() {
    const { register, formState: {errors}, handleSubmit } = useForm();
    const onSubmit = (data,e) => {
        console.log(data);
        //e.target.reset();
    }
```

Figura 58: Propietats especifiques de react-hook-form Font: Elaboració pròpia

Propietats com *...register()""{required:{value:true, message:}*, son pròpies de React. A més a més, el propi formulari et munta una resposta de tipus objecte emmagatzemada a

data. Addicionalment, el formulari de react et permet incloure diverses funcionalitats declarant-les just abans del *useForm*().

També, React ens permet aplicar lògica en el mateix fitxer, noteu que en la següent imatge els camps 2 i 3 estan desactivats fins que no triem el primer.

Un cop compilat i generat, ens mostra el següent:

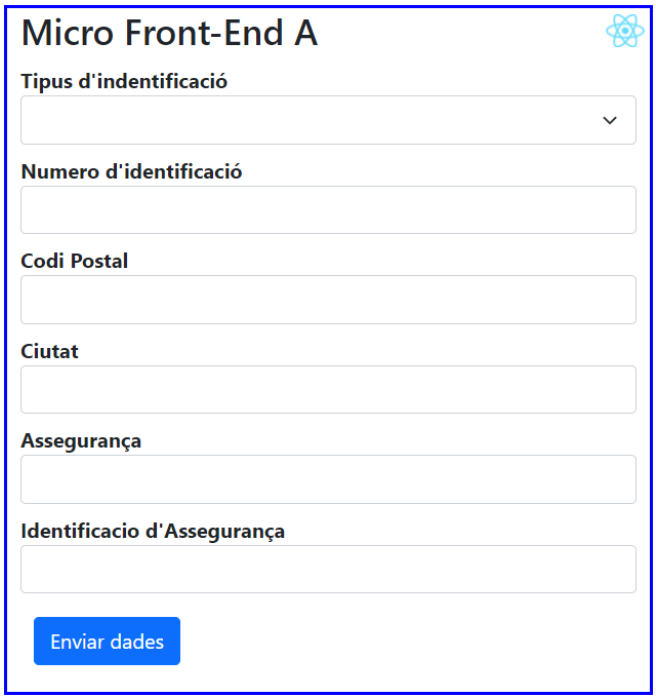


Figura 59: Diseny del component React acabat

- Preparació de React amb suport de Single-SPA

En una primera instancia hem de crear un projecte fora de l'aplicació Shell. Ho realitzarem utilitzant aquesta comanda:

create-single-spa –framework react

Durant la fase de creació, així com la contenidora, ens preguntarà un seguit d'opcions. Les seleccionem al nostre gust excepte el nom de la organització, que serà la referencia per a poder fer els imports a posterior. Aquesta referencia serà *tfmpoc*. L'arquitectura d'aquest projecte serà el següent:

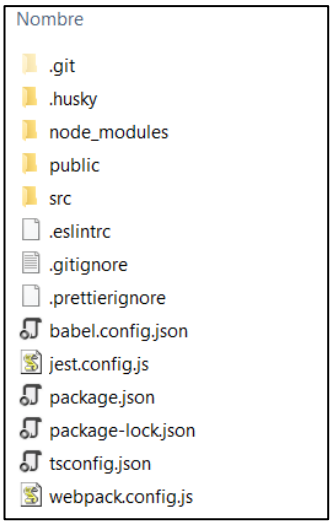


Figura 61: Estrucura amb suport de Single-SPA Font: Elaboració pròpia

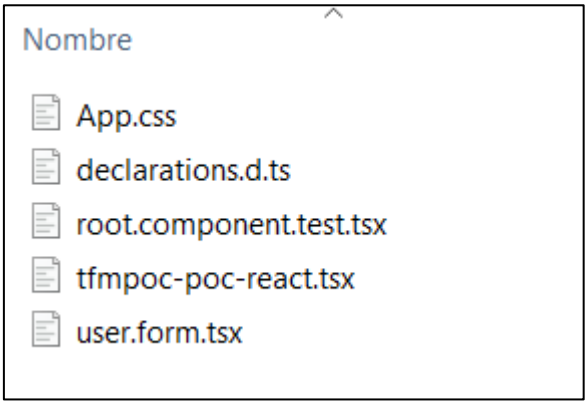


Figura 60: Estrucutura de src amb suport de Single-SPA Font: Elaboració pròpia

User-form.tsx → Contindrà el component que volem rendereritzar, a més a més React ens permet aplicar-li lògica al mateix arxiu.

Tfmpoc-poc-react.tsx → Fitxer que conté les funcions obligatòries de Single SPA per a poder usar el component a l'aplicació contenidora (mount, ummount i bootstrap).

```
const lifecycles = singleSpaReact({
  React,
  ReactDOM,
  rootComponent: HookForm,
  errorBoundary(err, info, props) {
    // Customize the root error boundary for your microfrontend here.
    return null;
  },
});

export const { bootstrap, mount, unmount } = lifecycles;
```

Figura 62: tfmpoc-react.tsx, amb les funcions de Single-SPA Font: Elaboració pròpia

En la imatge 67 ja estem preparant a la comunicació que viatjarà pel front. Declarem un Observable, tal i com fem al punt 6.1.2 del TFM, que publicarà la informació que l'usuari entrarà. Aquesta informació podrà ser recollida amb la referencia *reactdata*.

```
const observable = new Observable('reactdata');

function HookForm() {
  const { register, formState: {errors}, handleSubmit, reset} = useForm();
  const onSubmit = (data,e) => {
    if(Object.keys(data).length > 0){
      reset();
      observable.publish(data);
    }

  }
```

Figura 63: Lògica del formulari amb WindowedObservable Font: Elaboració pròpia

Un cop tot l'escenari preparat hem d'aixecar la instància a un port diferent que totes les altres aplicacions que vulguem incloure. Per aquesta aplicació la deixarem per defecte, ja que es el port 8080 i no influeix amb cap altre. Aixequem la instancia perquè es mostri a l'aplicació principal amb:

npm run start

I si volem treballar en mode local a l'hora que a la principal:

npm run start:standalone

A continuació detallem com incorporar Angular, probablement el framework que dona més problemes.

5.2.4 Angular

- Preparació d'Angular sense suport de Single-SPA

Per Angular també necessitarem Node.js. Un cop instal·lat procedirem a la instal·lació d'Angular. Instal·lem la línia de comandes (Command Line Interface) amb:

npm install -g @angular/cli

Amb aquesta comanda tindrem l'ultima versió d'Angular.

Per a la creació d'un nou projecte, utilitzem (my-app serà el nom de l'aplicació que creem).

ng new my-app

A continuació mostro el sistema de fitxers que tenim amb aquesta comanda:

Nombre	Fecha de modificación	Tipo	Tamaño
.git	20/12/2021 22:30	Carpeta de archivos	
node_modules	20/12/2021 22:40	Carpeta de archivos	
src	20/12/2021 22:29	Carpeta de archivos	
.browserslistrc	20/12/2021 22:29	Archivo BROWSER...	1 KB
.editorconfig	20/12/2021 22:29	Archivo EDITORCO...	1 KB
.gitignore	20/12/2021 22:29	Documento de tex...	1 KB
angular.json	20/12/2021 22:39	JSON File	4 KB
karma.conf.js	20/12/2021 22:29	Archivo JavaScript	2 KB
package.json	20/12/2021 22:39	JSON File	2 KB
package-lock.json	20/12/2021 22:39	JSON File	1,021 KB
README.md	20/12/2021 22:29	Archivo MD	2 KB
tsconfig.app.json	20/12/2021 22:29	JSON File	1 KB
tsconfig.json	20/12/2021 22:37	JSON File	1 KB
tsconfig.spec.json	20/12/2021 22:29	JSON File	1 KB

Figura 64: Estructura d'Angular Font: Elaboració pròpia

Els fitxers/carpetes que haurem de tenir en compte en aquesta prova són els següents:

- *Node_modules* → Carpeta on van totes les dependències instal·lades a partir de NPM.

- *Src* → Components, Routing i lògica de tota la nostra aplicació. Es on nosaltres crearem els components i implementarem la lògica d'angular.
- *Angular.json* → Proporciona valors predeterminats de configuració per tot l'espai de treball i de configuració especifica de tot el projecte.
- *Package.json* → És el cor del projecte. Guarda metadata important, defineix atributs funcionals per NPM, defineix scripts i identifica l'entry point del teu projecte.
- *Tsconfig.json* → Especifica els fitxers de l'arrel i les opcions del compilador que es necessiten.

Si entrem dins de *scr*, veiem la següent organització:

Figura 65: Estructura de src d'Angular Font: Elaboració pròpia

Aquí el més important és l'index.html, que és el primer fitxer que mostrarà el nostre navegador, aquí simplement hi ha col·locada una etiqueta html que referència al contingut que hi ha dins d'*app*

A assets hi col·locarem multimèdia vària. I a styles.css conte els estils globals de tota l'aplicació.

Si ens endinsem a app, ens trobem amb aquesta jerarquia:

Figura 66: Carpeta app d'Angular

Aquí aniran tots els components i la lògica de cadascun d'aquests. Per defecte, tots descendiran al component *app.component.html/app.component.ts,* però, que significa cada una d'aquestes extensions?

Els .component.ts conté tota la lògica del component, es comunica directament amb el .html que es on es pica totes les etiquetes html i es dona forma al component. El component.css defineix els estils propis dels components. *Module.ts* s'encarrega dels

imports generals del component (per les funcionalitats del component) i de diferents declaracions dels fitxers del component. Per una altra banda, *app-routing.module.ts* és un fitxer que poden ficar addicionalment si volem fer ús de Routing controlat dins del teu propi component (gestió de links). Per últim, fitxer de testing del component: *app.component.spec.ts*.

A continuació mostro el workflow de com funciona Angular:

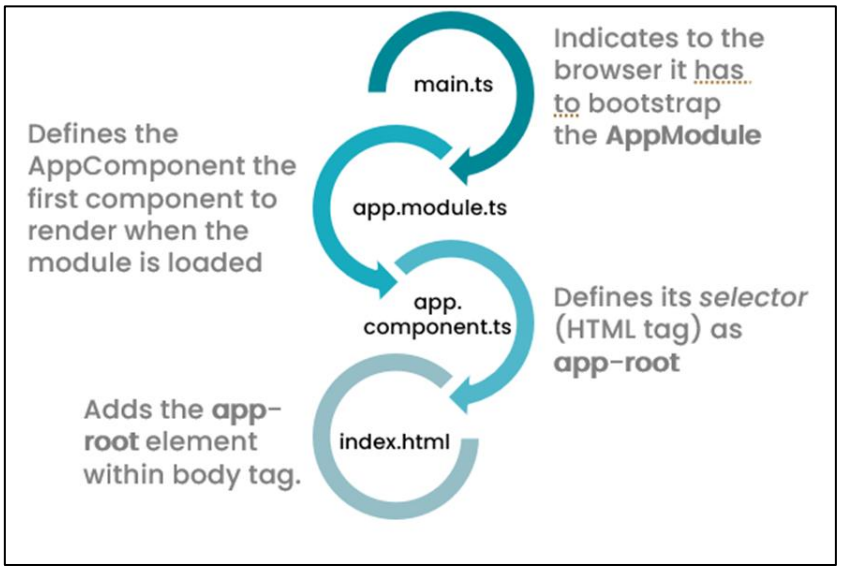


Figura 67: Workflow d'Angular Font: Elaboració pròpia

Un cop explicat el funcionament d'Angular per sobre, procedim a la creació del component.

Com ja hem dit a l'inici, la temàtica dels nostres components seran sobre els passos que ha de fer un usuari per introduir dades personals i dels detalls del seu vehicle. Finalment un component resum de tot el que ha anar seguint l'usuari.

Per fer-ho real, hem utilitzat funcionalitats específiques del framework d'Angular. Això ho hem fet, com a React, per veure si realment una aplicació que s'introdueix dins d'una altra és capaç de llegir propietats pròpies d'un entorn independent. El component llueix de la següent manera:

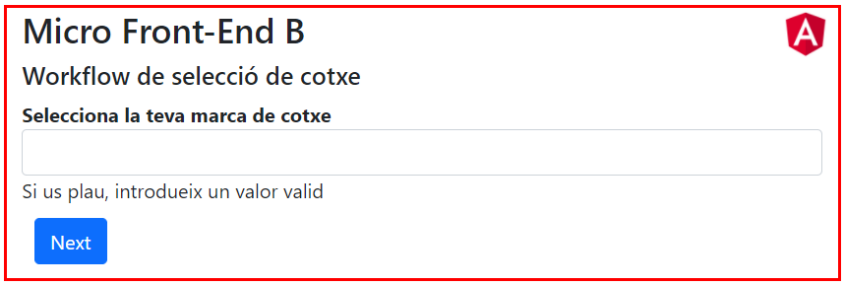


Figura 68: Component tipus Workflow amb Angular Font: Elaboració pròpia

En l'àmbit tècnic podem apreciar el codi HTML d'aquest component en la següent imatge:

```
<form [formGroup]="createForm">
    <div>
        <label><b>Selecciona el teu model de cotxe</b> ({{brand}})</label>
        <select class="form-control" [(ngModel)]="selectedOption" formControlName="carmodel" required>
            <option value=""></option>
            <option *ngFor="let model of models">{{model.name}}</option>
        </select>
        <span class="invalid-message" *ngIf="!createForm.get('carmodel').valid && createForm.get('carmodel').touched">
            {{requiredMessage}}
        </span>
    </div>
    <button class="btn btn-primary" (click)="goToBrandSelection()" id="but1">Back</button>
    <button class="btn btn-primary" (click)="SendData()" id="but1">Enviar Dades</button>
</form>
```

Figura 69: HTML amb propietats d'Angular Font: Elaboració pròpia

Com veiem en la imatge, fem ús de directives com: *formGroup, (ngSubmit), *ngIf*. Això ens dona la "pista" de que s'està utilitzant llenguatge propi d'Angular. Si ens fixem en la lògica que utilitzem per a aquest formulari, encara és més evident:

App.css U | brand-selection.component.ts U

PocSingleSPA > poc-angular > src > app > brand-selection > brand-selection.component.ts > BrandSelectionComponent > goToModelS

```
18
19
20  constructor(private api: ApiService, private router: Router) { }
21
22  get carbrand(): FormControl{
23    return this.createForm.get("carbrand") as FormControl;
24  }
25
26
27
28  ngOnInit(): void {
29    this.buildForm();
30    this.api.buildForm();
31    this.api.getBrands().subscribe((response: any) =>{
32      this.brands = response;
33      console.log(response);
34    });
35
36  }
37
38  buildForm(){
39    this.createForm = new FormGroup({
40      carbrand: new FormControl("", Validators.required)
41    })
42  }
43
44  goToModelSelection(){
45    if(this.createForm.valid){
46      this.api.selectedBrand = this.selectedOption;
47      this.api.GeneralForm.patchValue({carbrand: this.createForm.value['carbrand']});
48      this.router.navigate(['/model']);
49    }
50    else{
51      this.createForm.markAllAsTouched();
```

Figura 70: Lògica d'Angular Font: Elaboració pròpia

Amb tot això tindríem uns passos bàsics de creació de components i maneig d'Angular, a continuació exposo com fer un projecte amb Angular amb el suport de Single-SPA.

- Preparació d'Angular amb suport de Single-SPA

El primer pas es crear un projecte d'Angular de manera habitual:

ng new app

Quan ja el tinguem creat és ara quan li afegim el suport de Single-SPA per angular amb la comanda:

ng add single-spa-angular

Això ens afegira varis arxius d'interès a la nostra aplicació d'angular, però no deixa de ser el mateix que crea automàticament a React amb la comanda que hem explicat amb anterioritat.

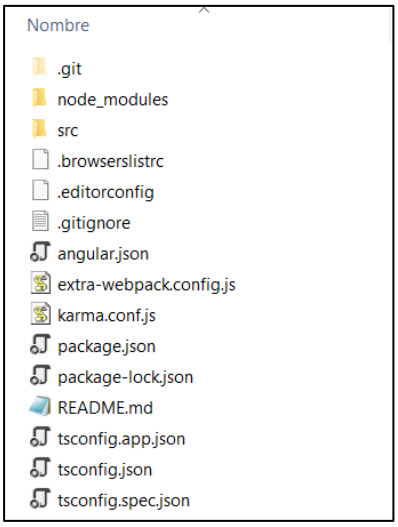


Figura 72: Root del projecte Angular amb suport Single-SPA Font: Elaboració pròpia

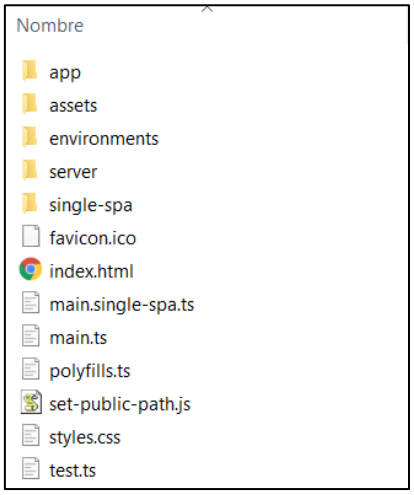


Figura 71: Carpeta src d'Angular amb suport Single-SPA Font: Elaboració pròpia

Un dels arxius que hem de tenir en compte és main.single-spa.js, dins de la carpeta src. Aquest és el que contindrà les funcions de *mount, ummount i bootstrap.*

Aleshores com que angular és molt independent, amb aquestes configuracions no seria suficient per donar-li l'ús que volem, encara hem de fer algunes configuracions:

A *app.module.ts* global el modifiquem de la següent manera:

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    RouterModule
  ],
  providers: [[{ provide: APP_BASE_HREF, useValue: '/poc-angular' }]],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Figura 73: Mètode per definir una rutar al MFE d'Angular Font: Elaboració pròpia

Fixem-nos sobretot amb l'atribut *providers*, posem aquests paràmetres per indicar a Angular on ha de renderitzar tot el seu contingut, és a dir a /poc-angular. Això ens provoca una limitació que avui dia no hem sabut resoldre: només podrem tenir Routing a un sol component en una mateixa pantalla i aquest serà angular.

Ara només hem definit la ruta de render, però hem de "publicar" tota aquesta informació per a que l'aplicació Shell pugui obtenir-la. Per a fer-ho ens ajudem d'un plugin anomenat *webpack-system-interop*:

npm install systemjs-webpack-interop -S

Per a fer ús d'ell, creem un arxiu qualsevol amb javascript i li fiquem les següent línies:

```
import { setPublicPath } from "systemjs-webpack-interop";
setPublicPath("@tfmpoc/poc-angular");
```

Figura 74: Configuració amb el plugin System-Webpack-Interop Font: Elaboració pròpia

És aquest arxiu el que voldrem importar al *main.single-spa.ts*.

Abans de poder aixecar la instància, si volem que es mostri el component hem de fer un petit canvi al Shell: Hem d'incorporar el concepte de Zones. Les zones serveixen per crear un context d'execució que ajuda als desenvolupadors a interceptar dades i tenir control de les operacions asíncrones. Sense l'import de l'Script de Zone-.js Angular no és capaç de mostrar-se al Shell. La solució implica incloure aquesta línia al *index.ejs*:

```
<script src="https://cdn.jsdelivr.net/npm/zone.js@0.10.3/dist/zone.min.js"></script>
```

Figura 75: Configuració de l'script Zone.js a la Shell Font: Elaboració pròpia

Amb tots aquests canvis ja podem aixecar la instància i, si hem importat aquesta última ruta que es mostra en el *webpack-system-interop* a l'aplicació Shell, ja podrem veure el component renderitzat.

Per aixecar la instància utilitzem aquesta comanda, que també té les configuracions adequades perquè tot funcioni correctament.

```
"serve:single-spa:poc-angular": "ng s --project poc-angular --disable-host-check --port 4201 --live-reload false"
```

Figura 76: Particularitat a l'hora d'aixecar aquesta instància Font: Elaboració pròpia

Això seria tot per fer ús d'Angular coexistint amb una altra aplicació amb altres frameworks alhora. Com ja hem comentat, ens ofereix la limitació de només poder ficar una aplicació tipus Angular coexistint a la mateixa pantalla. A més a més, a primera vista també tenim la limitació d'executar aquesta aplicació en mode *standalone*, això significa que no podem visualitzar l'aplicació de manera independent alhora que ho fem a l'aplicació contenidora. Per a resoldre-ho hem de tractar amb un plugin anomenat *standalone-single-spa-webpack-plugin*. A continuació mostro el seguiment de la lògica per a comunicar els MFEs.

```
SendData(){
  this.api.GeneralForm.patchValue({carmodel: this.createForm.value["carmodel"]});

  if(this.createForm.valid && this.api.GeneralForm.valid){

    //const newdata = JSON.stringify(this.api.GeneralForm.value)
    this.createForm.reset();
    observable.publish(this.api.GeneralForm.value);
    this.api.GeneralForm.reset();
    this.goToBrandSelection();

  }
  else{
    this.createForm.markAllAsTouched();
  }
```

Figura 77: Configuració amb Windowed-Observable a Angular Font: Elaboració pròpia

Recordem que son dos formularis que envien informació a una llista per front i que un formulari es connecta al *back* per saber que mostrar a un dels *selects*.

Un cop acabat amb Angular, passem a l'últim framework: Vue.

5.2.5 Vue

- Preparació de Vue amb suport de Single-SPA

Per Vue seguim el mateix procediment que a React, creem un nou directori on es destinarà tot el projecte de Vue i executem la següent comanda:

create-single-spa –framework vue

Com en els altres frameworks mostrem també l'arquitectura del que ens crea aquesta comanda:

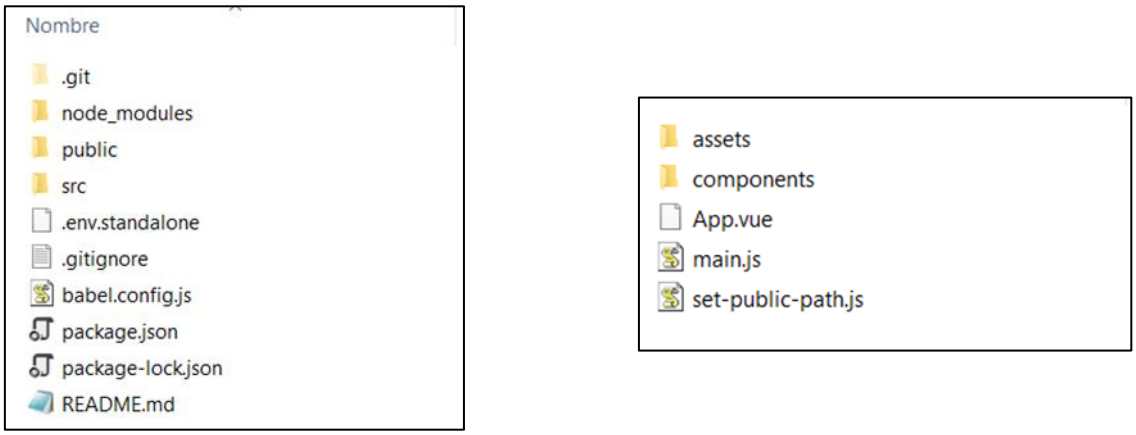


Figura 78 Sistema de fitxer de Vue amb suport Single-SPA Font: Elaboració pròpia

A main.js dins de src és on es troben les funcions de *mount, unmount i bootstrap*, les necessàries per poder treballar amb Single-SPA i el tema de les parcel·les.

App.vue és l'arxiu que fa de root per a tots els component que vulguem afegir, en el nostre cas, perquè no sigui un exemple molt simple, afegirem 2 component que podem veure a la següent imatge:

```
<template>
  <div class="container">
    <img class="img" alt="Vue logo" src="./assets/logo.png">
    <HelloWorld msg="Micro Front-End C"/>
    <DataTable/>
  </div>
</template>

<script>
import HelloWorld from './components/HelloWorld.vue'
import DataTable from './components/ListData.vue'

export default {
  name: 'App',
  components: {
    HelloWorld,
    DataTable
  }
}
</script>
```

Figura 79: Configuració d'App.vue a Vue Font: Elaboració pròpia

A Vue els components s'importen d'aquesta manera, com veiem hi ha un component on simplement es mostra un text a través d'una prop i un altre que simplement és una taula.

Un cop afegit els components que vulguem a Vue, hem de fer les connexions amb l'aplicació Shell, això significa repetir el pas d'Angular de publicar la ruta que hi haurà d'entendre la Shell.

Això ho fem de la següent manera:

1- Declarar un arxiu javascript qualsevol amb la següent informació al root del projecte.

```
import { setPublicPath } from "systemjs-webpack-interop";
setPublicPath("@tfmpoc/poc-vue");
```

Figura 80: Configuracio de l'arxiu que conté el plugin descrit Font: Elaboració pròpia

2- Instal·lar la llibreria systemjs-webpack-interop:

npm install systemjs-webpack-interop -S

Amb això ja es suficient per poder mostrar Vue a l'aplicació contenidora.

Pel que fa els components de Vue, hem utilitzat programació bàsica molt senzilla. Això sí, hem volgut respectar les funcionalitat pròpies del framework per afegir-li la complexitat de utilitzar funcions pròpies de cadascun i veure si son compatibles en l'entorn comú. A continuació mostro el component creat:

```
# App.css U   ListData.vue U ×
PocSingleSPA > poc-vue > src > components > ListData.vue > {} "ListData.vue" > script > default > mounted > observable2.subscribe() callback
<template>
  <div class="List">
    <div>
      <h5 class="text-left">Dades Usuari</h5>
      <ul class="text-left" v-for="(item,index) in DATAUSUARI" :key="item.id">
      <li> {{ item }} <br>  {{ INDEX[index*5+0] }} <br>  {{INDEX[index*5+1]}}  <br>  {{INDEX[index*5+2] }} <br>
        <br>  {{INDEX[index*5+4] }}</li>
      </ul>
    </div>
    <div>
      <h5 class="text-right">Dades Cotxe</h5    >
      <ul class= "text-right" v-for="item2 in DATACOTXE" :key="item2.id">
        <li> {{ item2 }} </li>
      </ul>
    </div>
  </div>
</template>
```

Figura 81: Programació a l'HTML de Vue Font: Elaboració pròpia

I a continuació la lògica del component amb la configuració de Windowed-Observable.

```
mounted(){
  const observable = new Observable('reactdata');
  const observable2 = new Observable('angulardata');
  observable2.subscribe((angular) => {this.DATACOTXE.push(angular.carbrand + " " + angular.carmodel)})
  observable.subscribe((react) => {this.DATAUSUARI.push("Tipus d'identificació: "+ react.idtype);
  this.INDEX.push("ID: "+ react.id ); this.INDEX.push("Codi Postal: " + react.zip);
  this.INDEX.push("Ciutat: " + react.city ); this.INDEX.push( "Assegurança: " + react.insurance );
  this.INDEX.push(  "Identificació d'Assegurança: " + react.idinsurance )
  });
```

Figura 82: Programació de la lògica de Vue amb Windowed-Observable Font: Elaboració pròpia

Amb això tindríem tota la infraestructura per poder aixecar totes les instancies alhora i poder veure els 3 MFE a l'aplicació principal! A continuació descrivim com s'ha fet la configuració del Back-End a Angular i després passem a veure els resultats finals!

5.2.6 Back-End

Per completar una mica més aquesta prova de concepte hem volgut fer-ho més real i incorporar un Back-End fet a mà. Per a fer-ho hem utilitzat la tecnologia *json-server*.

Json-server és una eina que et permet obtenir una API REST amb moltes facilitats i de manera molt practica. Ho hem fet servir a Angular per fer el formulari workflow dinàmic. Gràcies a això l'usuari aconseguirà opcions diferents que provenen del Back-End. Estaríem creant un formulari dinàmic.

Mostrem els passos per aconseguir aquest Back-End:

En primer lloc, instal·lem la tecnologia al nostre projecte amb la següent comanda:

npm intall -g json-server

Una vegada instal·lat, json-server és prou intel·ligent per a reconèixer arxius .json i tractar-los com si fos una base de dades o un Back-End. El que hem de fer, es crear aquest arxiu a on vulguem del nostre projecte. A continuació mostro el fitxer en qüestió:

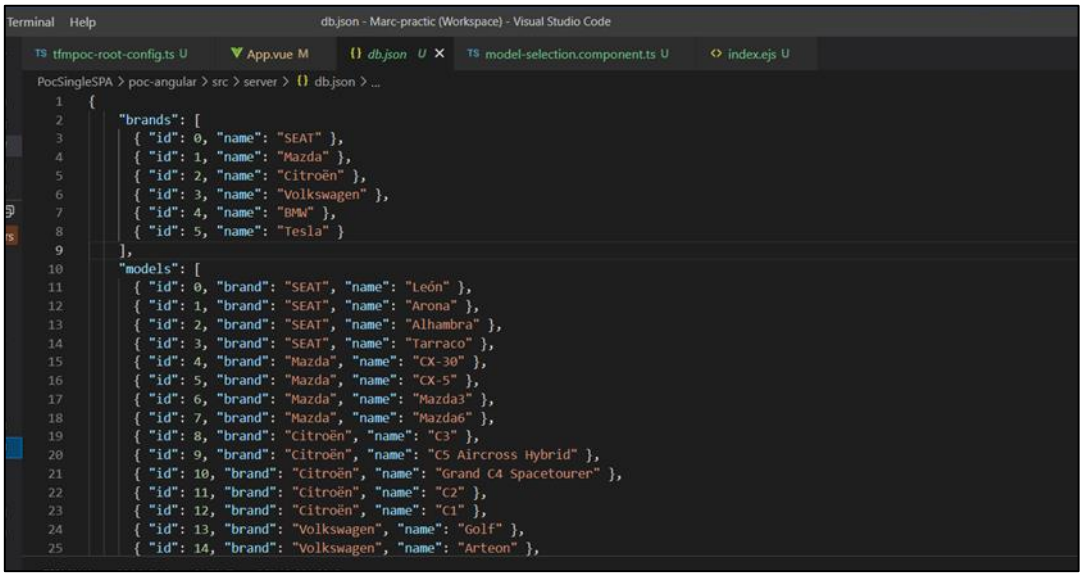


Figura 83: Fitxer db.json, encarregat de la construcció del Back-End Font: Elaboració pròpia

Per aixecar aquest entorn utilitzem la següent comanda:

json-server —watch db.json

Ara ja podem accedir a través d'HTTP a tota la informació que posem en el fitxer.json.

El que hem fet nosaltres és crear un arxiu de comunicació amb aquest Back-End amb totes les peticions que fem i després les importem al nostre projecte Angular.

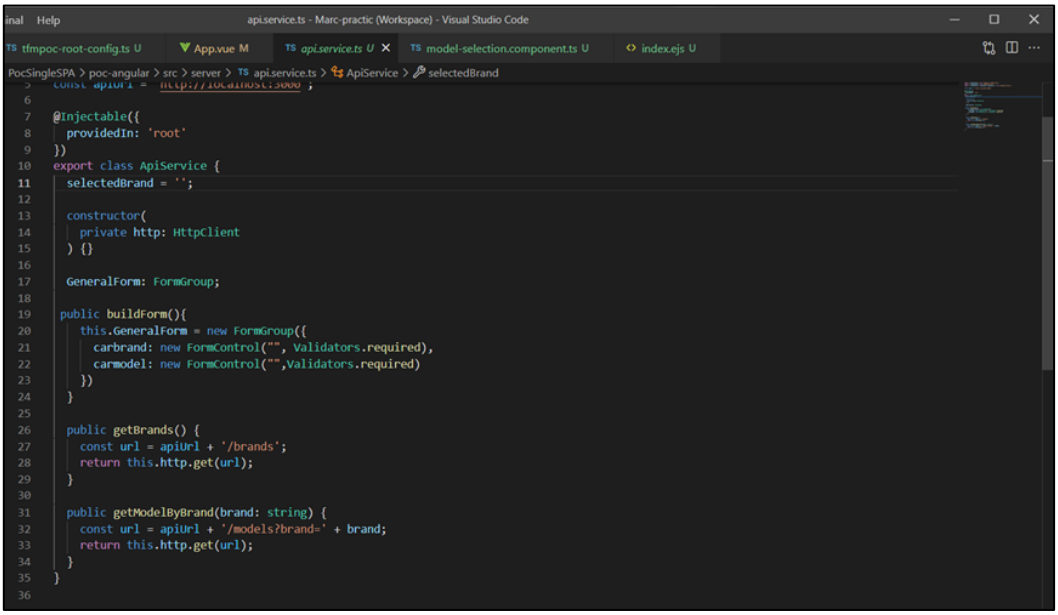


Figura 84: Api.service.ts, on controlem totes les connexions d'Angular Font: Elaboració pròpia

Així doncs, aquest seria tot el set-up per a fer una comunicació amb Front i Back utilitzant l'arquitectura dels MFE.

5.2.7 Resultat final

I aquí tenim el resultat final de la construcció amb aquest sistema. Com veiem, existeix una coexistència entre tres frameworks diferents en una mateixa ruta.

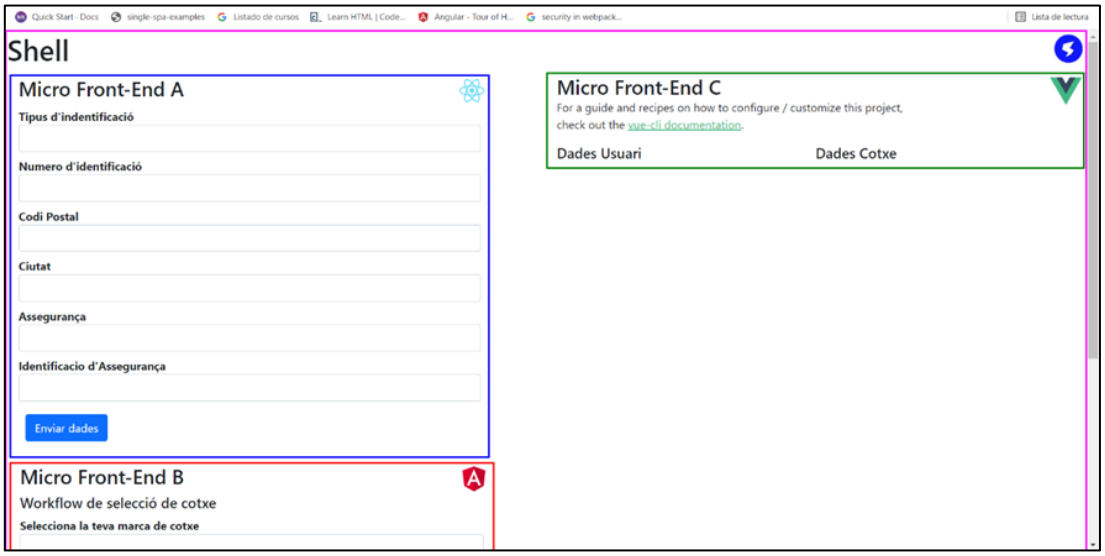


Figura 85: Resultat final visual Font: Elaboració pròpia

Amb aquesta infraestructura podem treballar de manera independent amb cadascun dels MFE.

Mostrem Vue:

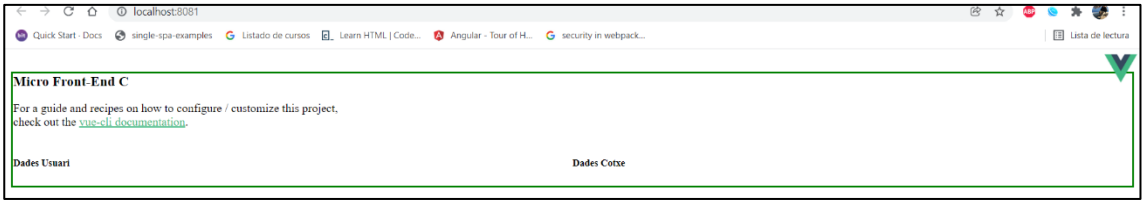


Figura 86: Vue.js de manera independent Font: Elaboració pròpia

Mostrem React:

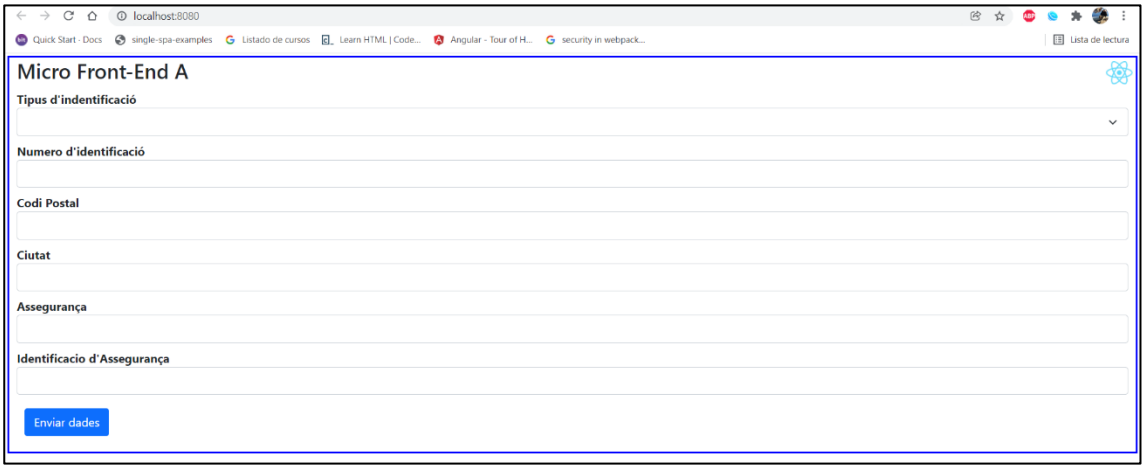


Figura 87: React.js de manera independent Font: Elaboració pròpia

Però no podem mostrar Angular, ja que ara per ara no hem aconseguit poder-lo mostrar de manera independent. Angular, al ser un framework amb moltes funcionalitats pròpies i tenir un ecosistema especial, s'han de fer unes configuracions addicionals amb el fitxer webpack que ve per defecte amb Single-SPA per poder-li donar aquesta opció, però per falta de temps no s'ha pogut treballar aquest apartat. Pot ser una molt bona practica poder editar Angular al nostre gust per poder treballar amb l'arquitectura dels MFE.

També mostrem la comunicació que existeix entre els frameworks en la següent imatge:

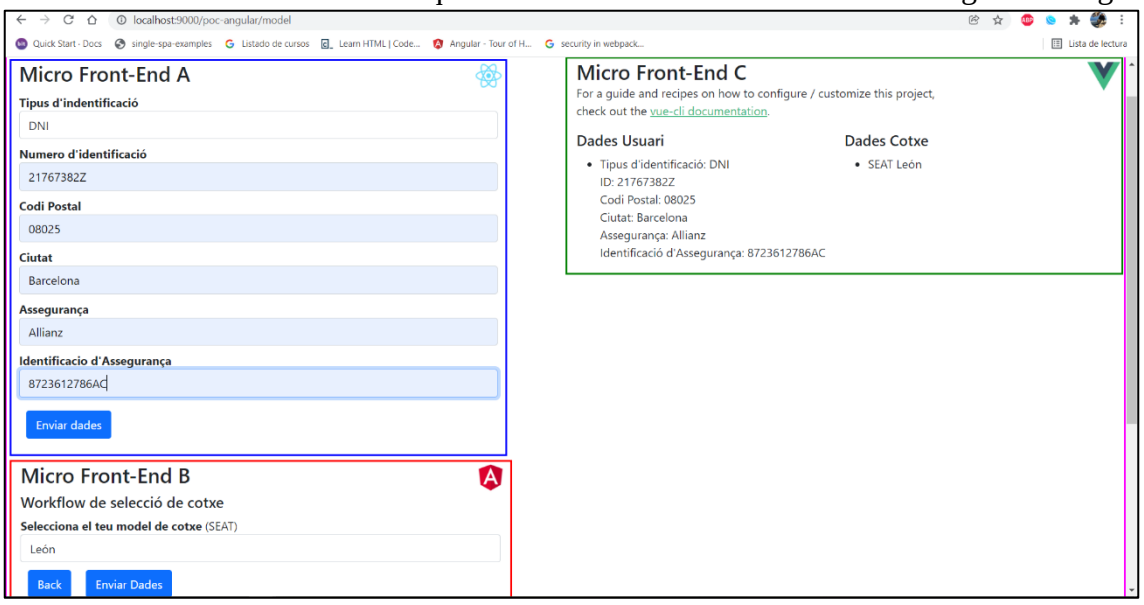


Figura 88: Comunicació entre els MFE utilitzant WO Font: Elaboració pròpia

Finalment també mostro com el Back-End està actuant en tota aquest arquitectura:

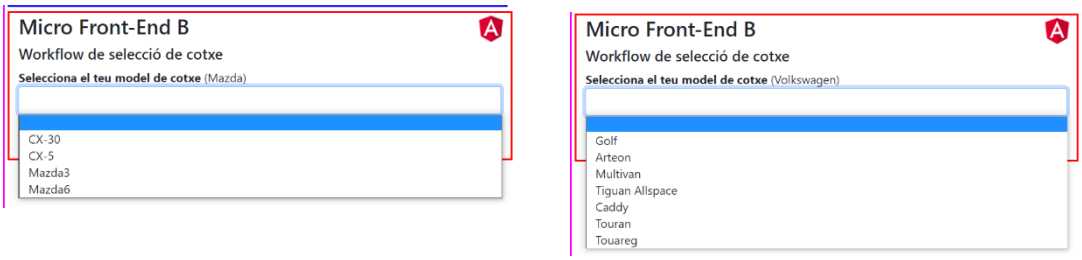


Figura 89: Back-End a Angular

En aquest cas només l'apliquem a Angular però es podria aplicar a qualsevol MFE d'aquesta arquitectura.

Amb això completem la integració dels MFE en un mateix entorn de producció. A partir d'aquesta proba de concepte podem escalar-la a projectes més grans i dedicar diversos equips a què treballin cadascuna de les tecnologies de manera independent.

5.3 Inconvenients de la integració

Durant la implementació de tot aquest procés han sorgit moltíssims obstacles que han fet que aquest projecte duri molt més de l'esperat.

A continuació exposo la primera proposta que vam començar a treballar i va acabar en una via morta. Durant la explicació aniré comentant tots els problemes que tenien cadascun dels passos.

- Primer escenari proposat

En una primera instància el sistema esquemàtic tindria una vista així:

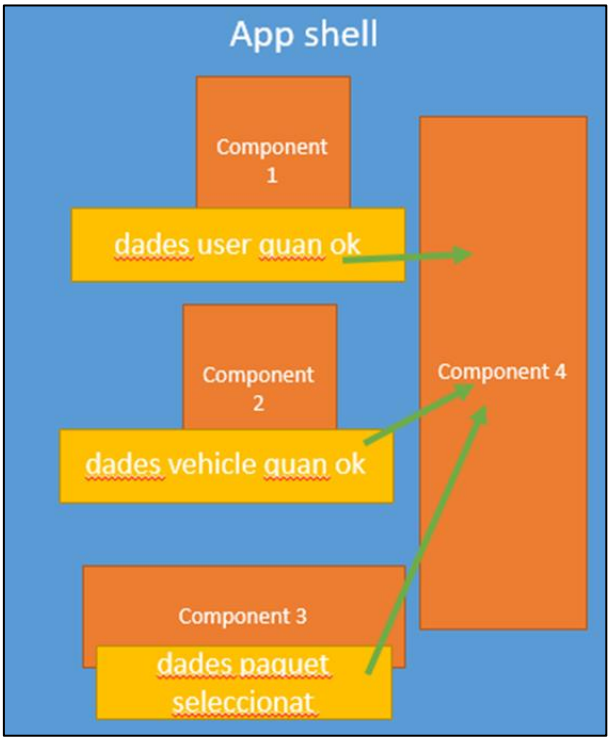


Figura 90: Esquema proposat inicialment Font: Elaboració pròpia

Com podem veure vindria a ser una aplicació amb quatre components on els tres primers es dedicarien a recollir dades de l'usuari i el quart faria el seguiment del que va triant l'usuari perquè tingui un resum de les seves accions.

La diferencia principal amb l'arquitectura que si funciona es el framework escollit: Module Federation en compte de Single-SPA.

A primera vista i havent tingut una primera presa de contacte amb tots els frameworks l'elecció més clara és Module Federation, ja que esta suportat per WebPack i totes les proves que hem fet han sigut satisfactòries i molt còmodes de realitzar. Podem veure les seves característiques i la puntuació proporcionades al apartat 5.2.9 del TFM.

- Explicació de perquè hem escollit aquesta arquitectura

Com ja hem comentat amb anterioritat, les tecnologies de Front que es voldrien utilitzar en aquesta implementació practica són les més demandades per l'empresa. Estem parlant de React, Angular i Vue.

Per demanda i perquè pensem que Angular és un ecosistema de programació Front molt més independent que React i Vue, triem que actuï com a contenidor. Així un cop creada l'instància contenidora, només s'haurà d'afegir els components que anem creant, sigui React, Vue o el mateix Angular.

A partir d'aquí muntem una arquitectura semblant a com ho fem a l'Annex I. Fem que Angular sigui l'aplicació que actuï de contenidora i React que actuï com un MFE.

Un cop tenim els components creats procedim a la integració d'aquests.

- Integració dels components a l'arquitectura MFE amb Module Federation

Per a aquesta fase, repetirem el mateix procés que vam utilitzar a la prova de concepte de Module Federation. Primer de tot instal·larem les dependències que necessita Webpack5 tant a Angular com a React.

yarn add webpack webpack-cli webpack-server html-webpack-plugin babel-loader webpack-dev-server css-loader

Després s'haurà de fer la configuració pertinent del arxiu webpack.config.js, aquesta configuració s'ha de fer a ambdues instancies com mostrem a continuació:

```
46  plugins: [
47    new ModuleFederationPlugin(
48      {
49        name: 'MFE2',
50        filename:
51          'remoteEntry.js',
52        exposes: {
53          './CarForm':
54            './src/cardetails-form',
55        },
56      }
57    ),
```

```
60  plugins: [
61    new ModuleFederationPlugin(
62      {
63        name: 'Shell',
64        filename:
65          'remoteEntry.js',
66        remotes: {
67          MFE2:
68            'MFE2@http://localhost:4201/remoteEntry.js',
69        },
70
71      }
72    ),
```

Figura 91: Configuracions de Webpack.config.js Font: Elaboració pròpia

Un cop feta aquesta connexió amb els arxius de webpack hem d'importar aquest component a l'aplicació d'Angular, per a fer-ho ens adonem que Angular ens ofereix una limitació molt important: El Routing.

Perquè Angular pugui mostrar components importats d'una altra aplicació ho hem de fer mitjançant rutes. No podem tractar-ho en l'àmbit d'aplicació com si fos un component propi d'angular. Aquí entra el concepte de l'Angular Routing.

Per aconseguir aquest mecanisme, Angular ens pregunta abans de la creació d'un projecte si volem que en contingui. Si li diem que si automàticament ens crea un arxiu anomenat *app-routing.module.ts*. Aquest té la següent pinta:

Figura 92: Configuració d'Angular Routing Font: Elaboració pròpia

Aquí definim un path a cadascun dels components que tingui la nostra aplicació i després mitjançant RouterLink a l'html podem referir-nos a aquest component que importem aquí.

Com veiem no reconeix "mf1/Carform" això és perquè angular no pot llegir informació que prové de webpack.config.js. Per "enganyar" al compilador, hem de crear un arxiu anomenat "decl.dl.ts" on allà declararem els mòduls que necessitem a l'arxiu de rutes i a qualsevol part de l'entorn d'Angular.

Quan ja tenim tota aquesta infraestructura muntada, només ens cal aixecar la instància mitjançant webpack. Això ho fem amb la comanda:

yarn webpack serve

És en aquest punt on ens adonem que alguna cosa no va bé...

Aquí el navegador ens començar a dir que no entén moltes de les extensions que li estem donant a angular: html de react, jsx de react, typescript d'angular... Per solucionar totes aquestes demandes del navegador importem loaders de webpack.

Webpack permet l'ús dels Loaders per preprocessar arxius. Això permet empaquetar qualsevol recurs estàtic més enllà de JS.

Quan ja solucionem tots els requeriments del navegador, aixequem ambdues instàncies i el mostrat és el següent:

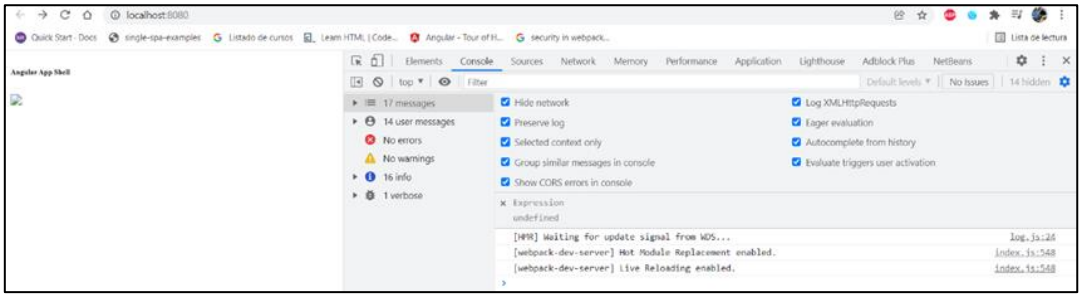


Figura 93: Incompatibilitat amb el que intentem renderitzar Font: Elaboració pròpia

Com veiem, l'aplicació s'aixeca sense cap mena de problema ni al compilador ni al navegador, però simplement només mostra el primer HTML d'angular, no aconsegueix indexar ni el component propi d'angular ni l'importat de React...

En aquest punt i provant diferents maneres amb aquesta versió d'infraestructura ens adonem que hem de tirar per una altra via amb Webpack Module Federation.

La següent prova realitza es fa seguint l'exemple de Manfred Stayer, un desenvolupador expert de Google treballant amb Angular i altres MFEs.

La seva versió consisteix amb el següent:

Manfred té una llibreria que podem instal·lar a angular anomenada *@angular-architects/module-federation*. Si instal·lem aquesta llibreria, automàticament angular ens crea un projecte configurat per utilitzar aquesta tecnologia:

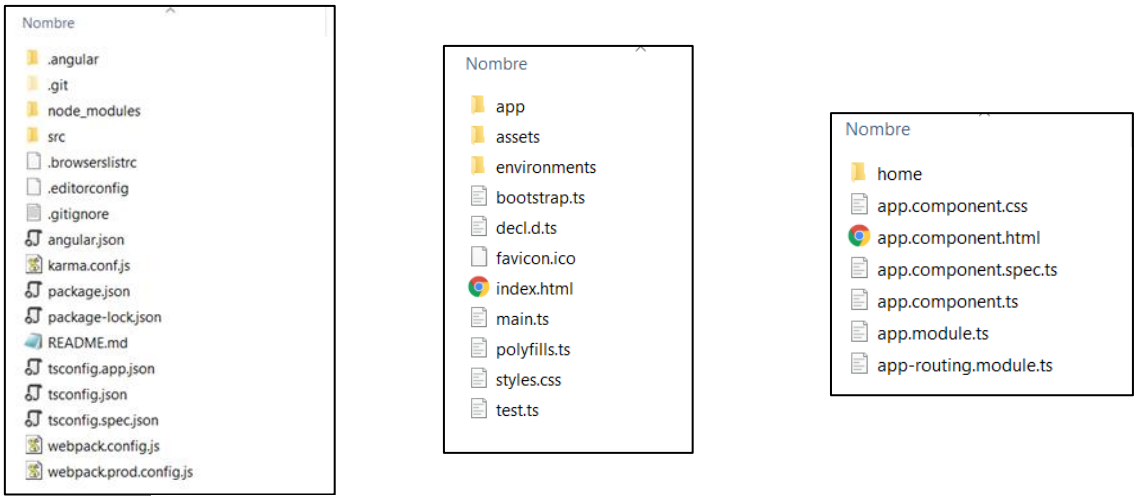


Figura 94: Estructura creada per la llibreria Font: Elaboració pròpia

Un cop tenint aquest projecte llest, només hem de fer unes poques configuracions per tenir-lo a punt.

LIMITACIÓ: Només podem utilitzar-ho amb 2 projectes Angular, no existeix un plugin ni una manera de poder adjuntar React i Angular actuant de Shell aquest últim des del moment de la investigació d'aquest projecte (novembre de 2021).

Sabent doncs que l'altra aplicació també serà d'Angular, l'únic que hem de fer és assignar a cadascuna de les aplicacions a un port diferent per poder-les executar en paral·lel (1), veure quins components exportem de l'aplicació que actuarà com a MFE (2), importar els component a l'aplicació que fa de Shell (3) i importar les rutes al projecte (4).

Associem els números d'aquest últim paràgraf al ordre de les imatges:

Figura 95: Configuració port d'Angular amb Webpack Font: Elaboració pròpia

Figura 96: Fitxer webpack a l'app Shell Font: Elaboració pròpia

Figura 97: Fitxer webpack al MFE Font: Elaboració pròpia

Figura 98: Configuració de rutes a Angular Font: Elaboració pròpia

Amb tot això configurat aixequem ambdues instàncies i en principi hauríem de poder veure una aplicació aixecada amb un enllaç per accedir al MFE. Però arribats a aquest

punt només ens trobem amb diversos problemes trets pel navegador sobre que no troba el fitxer de webpack del MFE, errors de sistema (*import.meta*) i un llarg etc...

Mirant de solucionar un altre cop tots els errors, en aquesta implementació vam estar moltes setmanes atrapat amb un bucle a la pàgina web on no podia aconseguir un altre cop el fitxer de webpack.

Addicionalment, vam provar altres mètodes partint d'aquest i l'únic que vam aconseguir va ser aquesta connexió durant uns moments, un cop refrescaves la pàgina no podies veure el teu MFE en condicions.

Aquí vam començar a plantejar-nos l'opció de què Module Federation amb Angular encara no estava massa madura i no era tan flexible ni "easy-to-use" com a React o a altres frameworks. Encara més se'ns plantejava la limitació de què Angular "no pot" assolir actuar com a Shell de cap altra tipologia diferent de framework i això per la nostra prova de concepte no complia amb les nostres expectatives. A més a més vam fer una altra prova de fer el pas revertiu actuant React com a Shell, però en aquest cas vam tenir ni el suposat problema de què React no sap entendre el app.module.ts d'Angular.

A continuació mostro l'única aplicació que vaig aconseguir amb aquesta tecnologia(Module Federation).

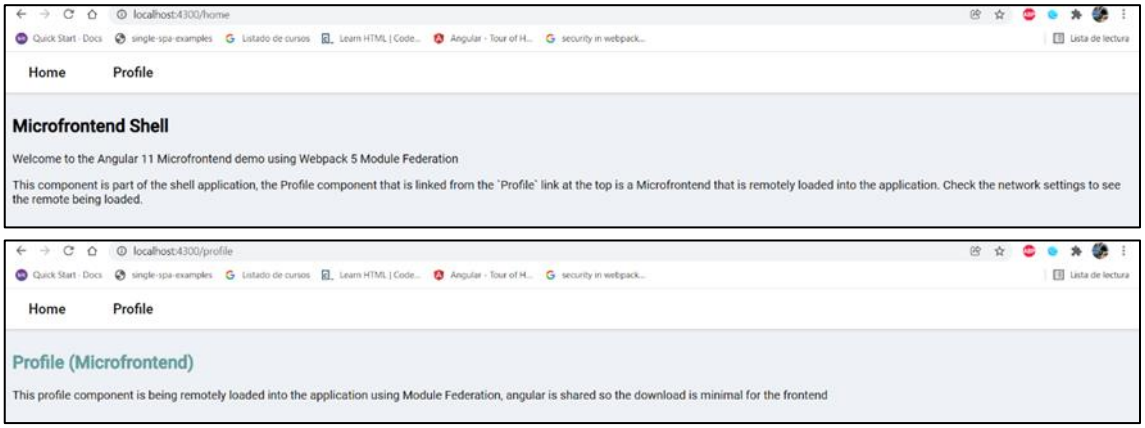


Figura 99: MFE inserit amb routing dins d'Angular Font: Elaboració pròpia

Com veiem aquí sí que hem aconseguit la unió d'una aplicació contenidora i una aplicació angular que actua com a MFE. El motiu de què això funcioni és la comanda que utilitzem per aixecar l'instància del MFE. Quan aixequem la instancià de la Shell, ho fem com normalment les posem en marxa amb Angular: ng serve. Però, amb l'altre utilitzem

El que estem fent aquí és executar dos scripts de node a la vegada. Per una banda, npm:build:watch, que ens permetrà compilar el nostre projecte per poder aplicar-li en temps reals canvis que desitgem, però no podrem accedir a ell a través d'un port

```
"watch": "concurrently npm:build:watch npm:http",
```

Figura 100: Comanda per aixecar la instància Font: Elaboració pròpia

independent. Per l'altra banda, utilitzem npm:http, que bàsicament aixeca un servidor HTTP independent perquè el nostre projecte estigui accessible per aquest canal (HTTP).

- Conclusions d’aquest escenari

Amb això resoldríem el problema de què Angular actues com a aplicació contenidora d'una altra. No obstant es queda força lluny d'aconseguir varis frameworks en una mateixa arquitectura de MFEs. A més a més amb angular tampoc hem vist la manera de què en una mateixa pantalla tinguem més d'una aplicació aixecada a la vegada. També, durant les fases de prova, hem fet el pas reversible de què React contingués Angular i tampoc tinguem èxit amb aquest pas.

Així que donem per incomplet aquest primer escenari i deduïm que Module Federation no té la suficient maduresa per complir una arquitectura com nosaltres la proposem.

6 Conclusions

En aquest apartat explicarem quines han estat les conclusions finals del projecte, els objectius assolits i les implicacions futures que tindrà.

La realització d'aquest projecte ha permès portar més enllà el concepte dels MFEs, una arquitectura molt recent per al desenvolupament amb entorns web. Aquest treball ha consistit en l'estudi de quatre tecnologies per a la creació d'aquesta arquitectura, l'estudi de mètodes de comunicació entre els diferents MFEs i les corresponents proves de concepte amb totes les tecnologies estudiades.

Single-SPA ha estat l'elecció final del framework més capaç per a la realització d'aquest projecte. S'ha fet una anàlisi exhaustiva de tres grans competències com són Module Federation, Piral i Bit. Single-SPA ens proporciona eines que els altres no com és la possibilitat de treballar amb diferents frameworks de programació Front-End en una mateixa pàgina, mòduls de suport per a moltes tecnologies i la facilitat de codi que ens ofereix.

Amb aquesta integració, obtenim la capacitat d'optimitzar els serveis webs obtenint així versatilitat en els nostres projectes de Front-End, dinamisme entre els equips que treballin per un mateix entorn i facilitat i rapidesa en la creació d'aquests.

Amb aquest projecte obrim la porta a una nova manera de fer feina en el món del desenvolupament web podent inclús integrar aquesta arquitectura a gestors de contingut com podrien ser AEM, Wordpress o Drupal.

Pels coneixements de desenvolupament web és necessari formar-se en els tres frameworks integrats a l'arquitectura MFE: React, Angular i Vue. Hem escollit aquests perquè avui dia són els més punters en el mercat i la més utilitzada per tots els desenvolupadors d'empreses d'arreu del món.

Les meves motivacions personals han estat el motiu d'elecció d'aquest projecte. He volgut sortir de la meva zona de confort per emergir-me en un àmbit completament nou, en una nova manera de fer feina com són els MFE. El meu interès va augmentar considerablement quan em van confiar el desenvolupament d'un projecte en fase d'experimentació i jo podria portar-lo a nivell on mai ningú abans havia estat.

De cara al futur he proporcionat un manual i unes eines pels equips de l'empresa per desenvolupar aplicacions pels seus clients. Caldrà doncs combinar les nocions que tinguin sobre el desenvolupament Front-End i incorporar el mètode desenvolupar a la secció pràctica d'aquest projecte. A més a més de tenir una explicació tècnica detallada i un punt de vista personal de l'autor del TFM. Hi ha altres expectatives de futur amb el mètode d'integració, entre les que destaco desenvolupar components específics per a cada client i fer seguiment i millores del mètode primari.

El fet que sigui un projecte innovador fa que la documentació d'arreu del món sigui molt escassa. Implica un treball d'investigació molt exhaustiu, en el qual s'han de mirar tot tipus de fitxers de configuració i entendre'ls per a poder fer un projecte d'aquest calibre.

Aquest projecte també va enfocat al meu entorn laboral, podent tractar temes del mateix àmbit i ajudar els equips que treballin sobre aquesta tecnologia. L'objectiu és seguir aprenent més sobre aquesta arquitectura de MFE, les tecnologies que la relacionen i, en un futur, escalar en aquest món de les tecnologies de TI amb la part de Front. Personalment, he adquirit nocions d'informàtica que desconeixia, ja sigui de Tecnologies de la Informació o d'Enginyeria del Software. Aquestes eines han fet que pugui seguir aprenent i desenvolupant-me com a Enginyer Informàtic a partir dels aprenentatges del grau i preparar-me pel la meva vida laboral imminent.

Aquest projecte també m'ha permès entendre el funcionament del món laboral, i m'he adonat que em falta molt per aprendre. La capacitat d'adaptació que tenen els equips davant els problemes i les solucions proposades han fet que aprengui profundament d'aquesta experiència. Inclús amb el desavantatge de la distància, com ha estat el cas d'aquest 2021 amb la pandèmia del Covid-19, m'he sentit un treballador més de l'empresa. Els meus companys i supervisors m'han proporcionat l'atenció que requeria i m'han acompanyat en els dubtes que sorgien; a més de posar-me en contacte amb altres sectors i especialistes per al desenvolupament de les meves tasques.

Respecte els serveis que donava la universitat i a les eines que m'han proporcionat per al desenvolupament del projecte, he après a documentar tots els objectius, processos i metodologies que he realitzat, amb la finalitat de proporcionar al meu departament una eina de qualitat.

7 Bibliografia

[1] Front-End. Pàgina d'Informació. Url: https://neoattack.com/neowiki/front-end/ [Consultat 1 d'Agost]

[2] Micro Front Ends. Pàgina Web. Url: https://micro-frontends-es.org/ [Consultat 18 de Juliol]

[3] NTTData. Web corporativa Url: https://www.nttdata.com/global/en/ [En actiu]

[4] Micro Serveis. Pàgina Web d'Informació https://microservices.io/ [Consultat 20 de Juliol]

[5] Allianz. Pàgina Web Official. Url: https://www.allianz.es/ [En actiu]

[6] SMART. Criteri de planificació. Url: https://en.wikipedia.org/wiki/SMART_criteria [Consultat 16 de Juliol]

[7] Github. The World's leading software development platform. Url: https://github.com [En Actiu]

[8] Google Drive. Almacenamiento en la nube. Url: https://www.google.com/intl/es_ALL/drive [En Actiu]

[9] Visual Studio Code. Code Editor. Url: https://code.visualstudio.com [En actiu]

[10] Json-server, creador de back-end's Url: https://www.npmjs.com/package/json-server [Consultat 10 de desembre]

[11] Swagger, creador de back-end's. Url: https://swagger.io/ [Consultat 13 de novembre]

[12] MockApi, creador de back-end's. Url: https://mockapi.io/ [Consultat 17 de novembre]

[13] Node.js, entorn per a desenvolupar Front-End. Url: https://nodejs.org/es/ [Consultat 15 de Juliol]

[14] Trello. Organitzador de projectes. Url: https://trello.com/es/home [En actiu]

[15] Module Federation. Pagina Web Oficial. Url: https://webpack.js.org/concepts/module-federation/ [Consultat 20 de Setembre]

[16] Angular, framework de Front-End. Url: https://angular.io/ [Consultat 1 de setembre]

[17] React, framework de Front-End. Url. https://es.reactjs.org/ [Consultat 13 d'Agost]

[18] Jira, eina de desenvolupament software. Url: https://www.atlassian.com/es/software/jira [Consultat 20 d'Agost]

[19] Confluence, eina de desenvolupament software.Url: https://www.atlassian.com/es/software/confluence [Consultat 20 d'Agost]

[20] Jenkins, eina de construcció de projectes. Url: https://www.jenkins.io/ [Consultat 20 d'Agost]

[21] One ERP. Software de gestió empresarial. Url: https://erp-one.com/ [Consultat 20 d'Agost]

[22] TechRadar, noticies de tecnologia a nivell mundial. Url: https://global.techradar.com/es-es [Consultat Juliol]

[23] API. Application Programming Interfaces. Url: https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces [Consultat 3 d'Agost]

[24]Conferencia Natalia Benditto. Url: https://www.youtube.com/watch?v=RJYEKVb2-38 [Consultat 5 d'Agost]

[25]Conferencia Manfred Steyer. Url: https://www.youtube.com/watch?v=vnBbtWKPEyM [Consultat 9 d'Agost]

[26] Blocks, Elements i Modifiers. Metodològia. Pàgina Web. Url: http://getbem.com/introduction/ [Consultat 25 Juliol]

[27] CSS-in-JS. Pàgina d'Informació. Url: https://octuweb.com/css-in-js/ [Consultat 2 d'Agost]

[28] Piral. Pàgina Web Oficial. Url: https://piral.io/ [Consultat 21 de Setembre]

[29] Single SPA. Pàgina Web Oficial. Url: https://single-spa.js.org/ [Consultat 1 d'Octubre]

[30] Bit Framework. Pàgina Web Oficial. Url: https://bit.dev/ [Consultat 5 de Setembre]

[31] Web Workers. Pàgina informativa. Url: https://developer.mozilla.org/es/docs/Web/API/Web_Workers_API/Using_web_workers [Consultat 27 de setembre]

[32] Custom Event. Pàgina informativa. Url: https://developer.mozilla.org/en-US/docs/Web/API/CustomEvent/CustomEvent [Consultat 29 de setembre]

[33] Props i callbacks. Pàgina informativa. Url: https://learn.co/lessons/react-using-callbacks-to-pass-information-lab [Consultat 30 de setembre]

[34] Windowed Observable. Referencia NPM. Url: https://www.npmjs.com/package/windowed-observable [Consultat 3 d'octubre]

--Documentació consultada per la realització de tot el projecte--

Creació d'un Back-End. Pàgina Web d'Informació. Url: https://developer.ibm.com/es/depmodels/microservices/patterns/create-backend-for-frontend-application-architecture/ [Consultat 22 de Juliol]

Eina d'Estils. Pàgina Web Oficial. https://sass-lang.com/ [Consultat 1 d'Agost]

Back-End. Pàgina d'Informació. Url: https://rafarjonilla.com/que-es/backend/#:~:text=El%20backend%20es%20la%20parte,la%20comunicaci%C3%B3n%20con%20el%20servidor. [Consultat 1 d'Agost]

Alguns mètodes de creació de MFE. Post. Url: https://sirshikher.medium.com/creating-microfrontend-application-using-bit-tool-why-only-backend-have-all-the-fun-of-16b7e591b7d8 [Consultat 15 d'Agost]

Creació de MFE amb Bit. Url: https://blog.bitsrc.io/how-we-build-micro-front-ends-d3eeeac0acfc [Consultat 4 Setembre]

GitHub amb exemple de Bit. Url: https://github.com/teambit/bit [Consultat 7 de Setembre]

CLI de Bit. Eina de creació. Url: https://harmony-docs.bit.dev/getting-started/initializing-workspace [Consultat 9 de sembre]

Estils a Bit. Url: https://docs.bit.dev/docs/best-practices#handling-styles [Consultat 11 de Setembre]

Seguretat a Module Federation. Url: https://webpack.js.org/guides/csp/#enabling-csp [Consultat 2 de Setembre]

Creació de MFE amb Module Federation. Url: https://blog.bitsrc.io/revolutionizing-micro-frontends-with-webpack-5-module-federation-and-bit-99ff81ceb0 [Consultat 5 de Setembre]

Exemples Github Module Federation. Url: https://github.com/webpack/webpack [Consultat 15 de setembre]

Móduls especials de Webpack. Url: [Consultat 17 de setembre] https://blog.jakoblind.no/css-modules-webpack/

Funcionament de Webpack. Paàgina d'informació. Url: https://www.arsys.es/blog/programacion/webpack-instalacion-funcionamiento/ [Consultat 18 de setembre]

Exemple de Piral. Github Oficial. Url: https://github.com/smapiot/piral [Consultat 22 de Setembre]

Com fer MFE amb Piral.Pàgina d'Informació Url: https://www.infoq.com/news/2021/06/piral-microfrontends/[Consultat 25 de setembre]

Exemples de MFE. Pàfina d'Informació https://dev.to/dantederuwe/my-experiences-creating-a-netflix-clone-using-microfrontends-1n46[Consultat 28 de setembre]

Seguretat a Single-SPA. Url: https://single-spa.js.org/docs/faq/#does-single-spa-require-additional-security-considerations [Consultat 3 d'Occtubre]

Tipus de MFE a Single-SPA https://es.single-spa.js.org/docs/module-types/ [Consultat 5 d'Octubre]

Exemples a Github de Single-SPA.https://github.com/single-spa/single-spa [Consultat 8 d'Octubre]

Estils a Single-SPA. Url: https://single-spa.js.org/docs/ecosystem-css/ [Consultat 9 d'Octubre]

Imports Dinàmics en la programació. Pàgina d'Informació. Url: https://blog.logrocket.com/speed-up-react-app-dynamic-imports-route-centric-code-splitting/ [Consultat 15 d'Octubre]

Angular amb Module Federation. Url: https://www.angulararchitects.io/en/aktuelles/the-microfrontend-revolution-part-2-module-federation-with-angular/ [Consultat 10 de novembre]

8 Annexos

8.1 Annex I: Proba de concepte de Module Federation

Començarem veient que pot fer Module Federation en aquesta primera presa de contacte.

Els primers passos per a l'ús d'aquesta tecnologia és tenir el gestor de paquets NPM i conseqüentment s'adquireix amb Node.js. Node és un entorn multiplataforma de temps d'execució per JavaScript amb el motor de V8 de Chrome. En poques paraules, necessari per a poder executar en temps real l'ús d'aquestes tecnologies, gestionar els seus paquets i poder operar amb elles.

El primer que s'ha de fer és crear com a mínim dues aplicacions independents que actuaran com els nostres MFE.

npx create-react-app mfe1

npx create-react-app mfe2

El següent pas es instal·lar les dependències que utilitzar WebPack, aquestes son necessàries per tot el tema de poder posar estils, utilitzar WebPack amb el CLI i demés característiques internes de l'aplicació. Aquest pas s'haurà de fer en tots els MFEs que tinguem.

yarn add webpack webpack-cli webpack-server html-webpack-plugin babel-loader webpack-dev-server css-loader

Un cop completat aquest pas, anem a un editor de codi i, en el meu cas, que utilitzo React com a creació d'aplicació canviem del fitxer index.js a les dues apps per bootstrap.js. Aquest pas es necessari perquè necessitem que index.js es carregui asincronament per esperar les accions de WebPack.

Ara el que hem de fer és crear un nou index.js que simplement faci un *import('./bootstrap').*

Un cop amb aquesta estructura ara sí que ja podem crear els nostres components dins la nostra aplicació.

Com que és una prova de concepte, l'aplicació A, anomenada MFE1 contindrà un botó i la B, anomenada MFE2 farà de contenidora afegint alguna característica.

En el nostre cas el component de l'aplicació A es alguna cosa tan senzilla com aquesta.

```
App.js M
PreEstudi > React+WP5+MF > MF > mfe1 > src > App.js > App
1  import React from 'react';
2  import Button from './Button';
3  function App() {
4    return (
5      <div>
6        <h1>MFE1</h1>
7        <Button>
8          {' '}
9          MFE1 Button
10       </Button>
11     </div>
12   );
13 }
14
15 export default App;
```

Figura 102: App.js contenint el component Font: Elaboració pròpia

```
Button.js U
PreEstudi > React+WP5+MF > MF > mfe1 > src > Button.js > ...
1  import React from 'react';
2
3  const Button = () => (
4    <button>MFE1 Button</button>
5  );
6
7  export default Button;
```

Figura 101: Button.js, conté el component botó Font: Elaboració pròpia

El següent pas i el més important, és la creació i la correcta configuració del fitxer webapck.config.js. Aquest és l'arxiu que gestionarà tot el tema del mòdul de Webpack i del plugin de Module Federation.

Aquest fitxer llueix de la següent manera a l'aplicació A.

```
webpack.config.js U
PreEstudi > React+WP5+MF > MF > mfe1 > webpack.config.js > <unknown> > module > rules
1  const HtmlWebpackPlugin = require('html-webpack-plugin');
2  const ModuleFederationPlugin = require('webpack/lib/container/ModuleFederationPlugin');
3  module.exports = {
4    mode: 'development',
5    devServer: {
6      port: 8083,
7    },
8    module: {
9      rules: [
10       {
11         test: /\.js?$/, /* The following line to ask babel to compile any file with extension.js */
12         /* exclude node_modules directory from babel. Babel will not compile any files in this directory*/
13         exclude: /node_modules/, // To Use babel Loader
14         loader:
15           'babel-loader',
16         options: {
17           presets: [
18             '@babel/preset-env' /* to transfer any advansed ES to ES5 */,
19             '@babel/preset-react',
20           ], // to compile react to ES5
21         },
22       },
23     ],
24   },
25   plugins: [
26     new ModuleFederationPlugin(
27       {
28         name: 'MFE1',
29         filename:
30           'remoteEntry.js',
31         exposes: {
32           './Button':
33             './src/Button',
34         },
35       }
36     ),
37     new HtmlWebpackPlugin({
38       template:
39         './public/index.html'
```

1 2 3 4

Figura 103: Fitxer Webpack detallat Font: Elaboració pròpia

El primer de tot és importar els plugins necessaris perquè funcioni l'aplicació amb WP5.

1. Aquesta és la primera línia, que genera un index.html que permetrà afegir els corresponents mòduls. I la segona línia, que importa Module Federation com a tal, que més endavant permetrà la comunicació entre les dues aplicacions independents.
2. El mode assignat és per evitar alguns errors en la fase de compilació i sobretot per indicar-li si volem una aplicació en fase de desenvolupament o de producció. A més a més també especifiquem el port on correrà aquesta aplicació un cop compilada.
3. Aquesta part és per fer-li saber al codi que va per darrere que permetem codi JavaScript i extensions de JSX. També li estem permetent la interpretació i la traducció amb React.
4. El plugin de MF com a tal. Conte 4 paràmetres. *Name, FileName ,Remotes i Exposes.* El *name* és per referenciar l'aplicació en qüestió. El *filename* és per indicar quin serà el fitxer de comunicació. *Remotes* serveix per poder referenciar a alguna aplicació i obtenir arxius d'aquella. I *Exposes* és per exposar de cara a les altres aplicacions què estàs disposat a compartir.

A l'aplicació B també haurem d'afegir aquest arxiu però amb unes petites modificacions.

La primera i la més obvia és el port:

```
module.exports = {
  mode: 'development',
  devServer: {
    port: 8082,
  },
```

Figura 104: Configuració del port a Webpack
Font: Elaboració pròpia

No podem operar la dues aplicacions diferents en el mateix port.

L'altre és la modificació de la secció del plugin.

```
plugins: [
  new ModuleFederationPlugin(
    {
      name: 'MF2',
      filename:
        'remoteEntry.js',
      remotes: {
        MFE1:
          'MFE1@http://localhost:8083/remoteEntry.js',
      },
    }
  ),
```

1 2

Figura 105: Configuració del plugin Module Federation Font: Elaboració pròpia

Com podem veure, aquí referenciem el nom de MFE1(2) i a continuació indiquem la direcció HTTP d'on es troba aquesta referencia. Fent això, estem indicant que a MFE1(1) "te" el contingut dels components exposats a l'aplicació A.

Així doncs nomes ens cal importar-lo en la nostra aplicació de la següent manera:

```
const MFE1_Button = React.lazy(
  () => import('MFE1/Button')
);
```

Figura 106:Import i referenciació del MFE extern Font: Elaboració pròpia

Amb aquest exemple tant senzill podem comunicar qualsevol tipus de components entre aplicacions independents!

Si volem posar en marxa aquestes aplicacions, utilitzem yarn amb el cli de webpack:

yarn webpack serve

El resultat d'aquesta senzilla proba de concepte seria el següent:

Al port 8083, Aplicació A(MFE1) tenim el botó:

Figura 107: MFE amb el botó

Al port 8082, Aplicació B (MFE2) tenim el contenidor amb el botó.

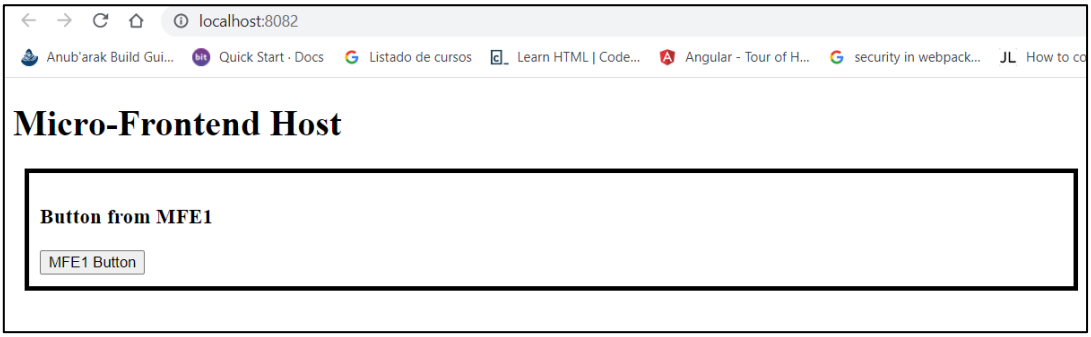


Figura 108: MFE2 amb MFE1 integrat Font: Elaboració pròpia

NOTA: Aquesta comunicació és possible perquè ambdues aplicacions estan corrent contínuament. Si donem de baixa l'aplicació A (la que té el botó) l'aplicació B no podrà renderitzar aquest contingut.

8.2 Annex II: Proba de concepte de Piral

Una vegada introduït Module Federation en aquesta prova de concepte, veiem que pot fer Piral, un altre framework molt interessant generador de portals webs desenvolupats amb MFEs. Piral, tanmateix també necessita els requeriments de node.js i el ja inclòs gestor de paquets npm.

El primer que farem és instal·lar tot el necessari per a poder tenir-lo en la nostra màquina personal amb la comanda:

npm i piral-cli

A continuació el que haurem de fer es crear l'aplicació contenidora o "Application Shell"

piral new –target my-app

En aquest cas, *my-app* serà l'aplicació que renderitzarà tots els MFE que haguem creat a partir d'aquesta. Per veure que hi conté aixequem l'instància amb: $ Piral debug.

Com ja hem comentat, Piral genera els MFEs o Pilets a partir de l'instància base. Per això el següent pas que hem de fer es la compilació d'aquesta instància. Això ho aconseguim amb:

piral build

Aquesta comanda generarà un arxiu .tgz a la carperta "dist". Aquesta serà la referencia que tindran els demés MFEs per a la seva pròpia creació. Per a crear-ne un utilitzarem el següent:

pilet new ./my-app/dist/emulator/my-app-1.0.0.tgz –target my-pilet

Un cop tenim la instància contenidora i el número de pilets que volem, podem començar a operar entre ells. En aquest exemple tenim una aplicació contenidora i 2 pilets. El primer registra una pagina amb text pla a la barra de navegació i l'altre també registra una pàgina però, que connecta amb una API per importar una llista d'elements.

La següent imatge mostra el Pilet A. Com podem veure utilitzar una pròpia API interna per la gestió del propi framework. Per això l'ús de *app.resgisterPage*, *app.registerMenu* i moltes més que no entrarem en detall.

Figura 109: Programàció bàsica del Pilet A Font: Elaboració pròpia

En la següent imatge tenim el Pilet B, que registra una pàgina en el menú i obté dades d'una API Externa.

```
TS index.tsx
my-pilet2 > src > TS index.tsx > ...
1  import * as React from 'react';
2  import { PiletApi } from 'my-app';
3  import { MyPageOfPosts } from './MyPage';
4  import { MyPageMenu } from './MyPageMenu';
5
6  const apiUrl = 'https://jsonplaceholder.typicode.com/posts';
7
8
9  export function setup(app: PiletApi) {
10   const connect = app.createConnector<Array<Post>>(() => fetch(apiUrl).then(res => res.json()));
11   app.registerMenu(MyPageMenu);
12   app.registerPage('/my-page', connect(MyPageOfPosts));
13 }
14
15
```

Figura 110: Programació bàsica del Pilet B Font: Elaboració pròpia

Però... com fa Piral per poder ficar aquestes dues aplicacions independents en una mateixa pàgina?

Bé, per a resoldre això tenim la següent eina → El Cloud de Piral

A traves d'aquest entorn, tu pots registrar un espai per a pujar traves del CLI els teus Pilets i poder-los mostrar tots de cop dins un portal web.

Al final té un aspecte així:

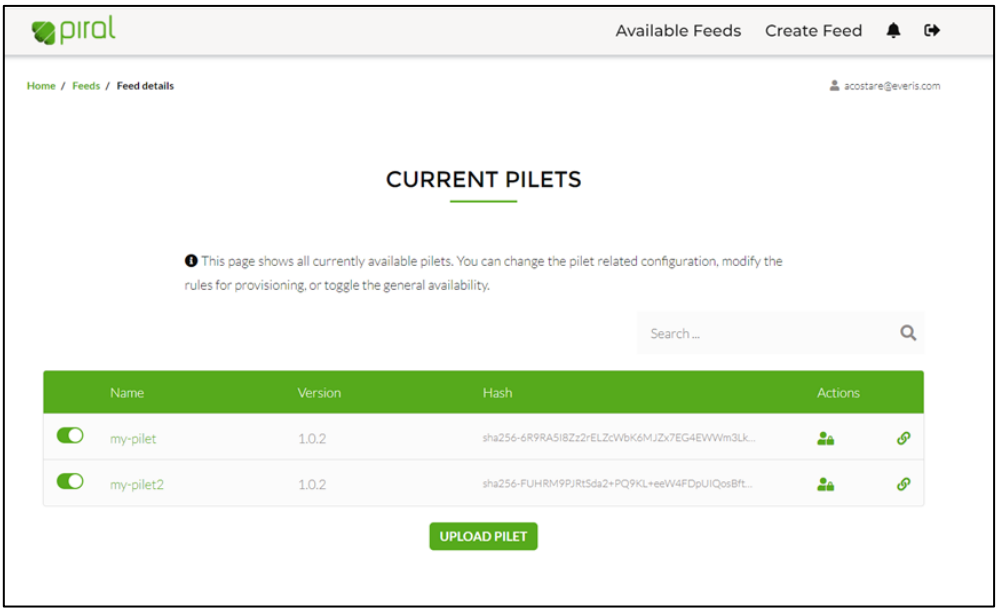


Figura 111: Entorn cloud de Piral amb el corresponents Pilets Font: piral.io

Per a poder-ho pujar cadascun del pilets, haurem d'anar al directori de cadascun d'ells i utilitzar la següent comanda

pilet publish –fresh—url https://feed.piral.cloud/api/v1/pilet/my-tutorial-feed --api-key <your-api-key>

L'URL consistirà amb la de la Feed creada dins del cloud de Piral. La podem trobar fàcilment a dins de la secció de feeds. L'API key només és general la primera vegada que crees una feed.

Un cop pujat ambdós pilets, només haurem d'anar al codi de l'aplicació que genera els 2 pilets i canviar l'URL per l'URL de la feed que hem creat.

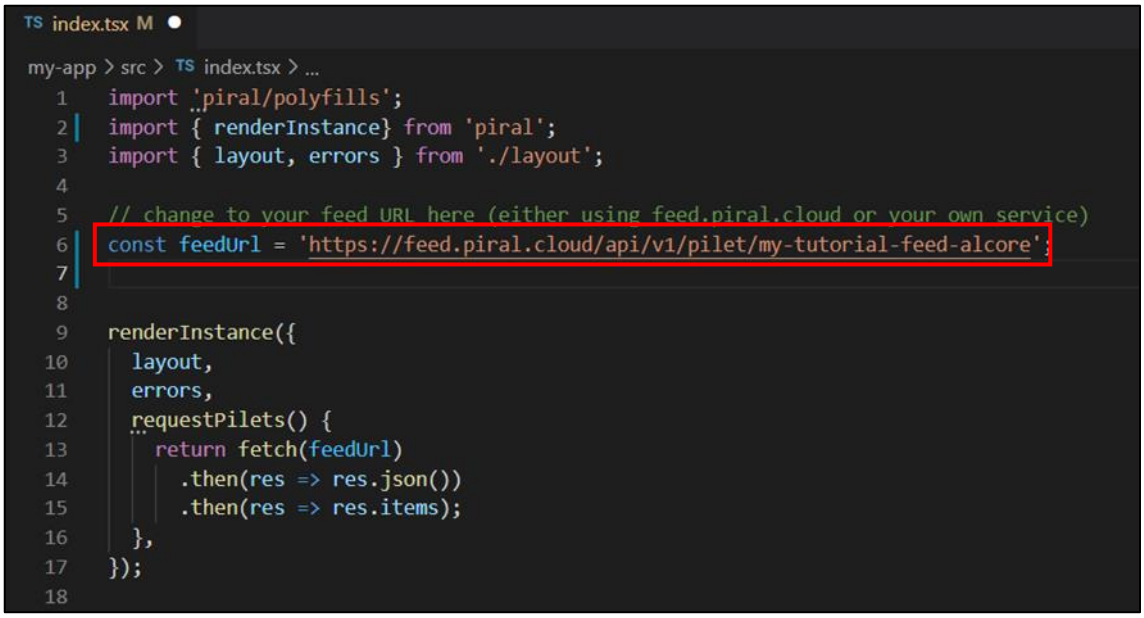


Figura 112: Ubicació d'on es defineix la URL de la pàgina base Font: Elaboració pròpia

El resultat final un cop fet tot això és el següent:

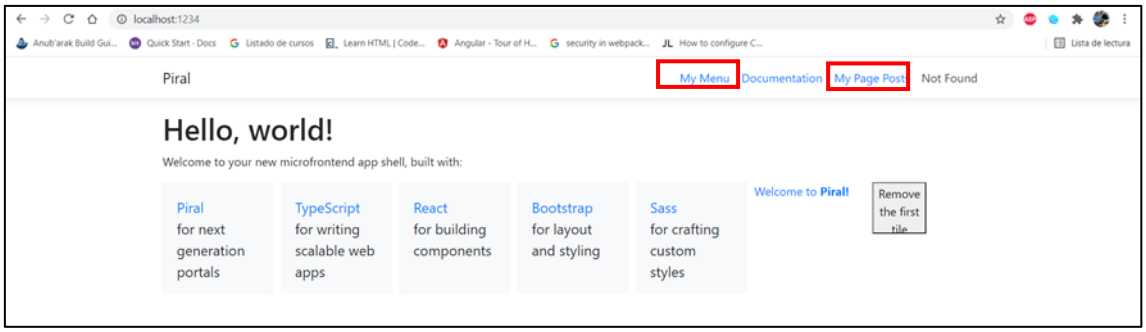


Figura 113: Resultat visual un cop treballat amb MFE a Piral Font: Elaboració pròpia

Veiem que a la barra de navegació esta el link "My Menu" que fa referencia al Pilet A. I "My Page Posts" que fa referencia al B.

8.3 Annex III: Proba de concepte de Single SPA

Una vegada introduït Module Federation i Piral veiem que pot fer Single-SPA, un altre framework molt interessant generador de portals webs desenvolupats amb MFEs. Single-SPA, tanmateix també necessita els requeriments de node.js i el ja inclòs gestor de paquets npm o yarn .

El primer que s'ha de fer és instal·lar *create single spa* amb la següent comanda:

yarn global add create-single-spa

A continuació executem el gestor de paquets instal·lat amb *npx create-single-spa*

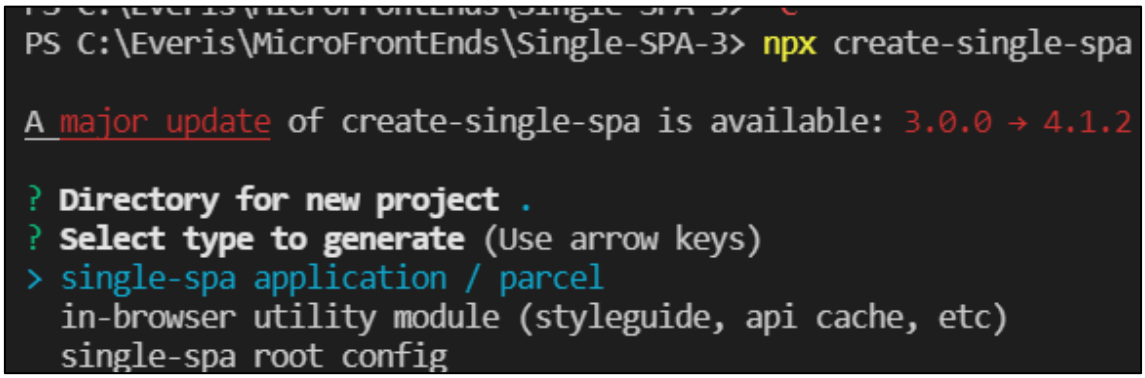


Figura 114: Seguiment de la instal·lació de Single-SPA Font: Elaboració pròpia

Ens sortirà una instal·lació guiada on ens preguntarà quin es el directori del projecte, el tipus d'aplicació, el llenguatge de programació i altres característiques per a la corresponent configuració.

Una vegada creada l'aplicació "single-spa root config", ja podem començar a crear les altres aplicacions independents, on al final del dia acabaran essent MFEs. Aquesta aplicació creada te una estructura com aquesta

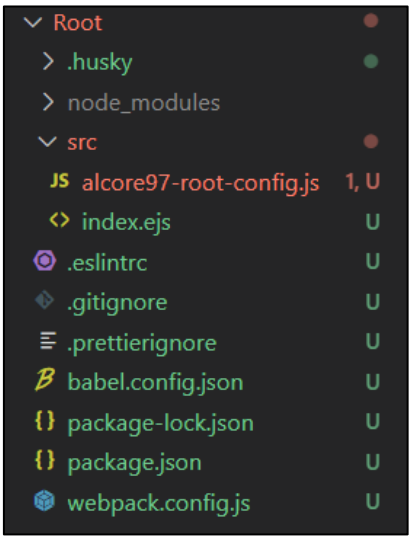


Figura 115:Estrucura de Single-SPA
Font: Elaboració pròpia

En aquest cas alcore97-root-config.js, és on renderitzarem totes les aplicacions (MFEs) que li vulguem donar al nostre portal web.

Index.ejs, és el punt d'entrada del sistema, aquí importarem les dependències dels frameworks que contindrà el nostre portal a més a més de la ruta HTTP de les nostres aplicacions independents. Tot això opera per darrere sobre el framework System.js, un dels frameworks que vam considerar com a possible estudi per la creació de MFEs.

```
<script type="systemjs-importmap">
  {
    "imports": {
      "single-spa": "https://cdn.jsdelivr.net/npm/single-spa@5.9.0/lib/system/single-spa.min.js",
      "react": "https://cdn.jsdelivr.net/npm/react@16.13.1/umd/react.production.min.js",
      "react-dom": "https://cdn.jsdelivr.net/npm/react-dom@16.13.1/umd/react-dom.production.min.js"
    }
  }
</script>
```

Figura 116: Imports a Single-SPA (Aplicacions) Font: Elaboració pròpia

```
<% if (isLocal) { %>
<script type="systemjs-importmap">
  {
    "imports": {
      "@alcore97/carousel": "//localhost:9002/alcore97-carousel.js",
      "@alcore97/header": "//localhost:9001/alcore97-header.js",
      "@alcore97/root-config": "//localhost:9000/alcore97-root-config.js"
    }
  }
</script>
<% } %>
```

Figura 117: Imports a Single-SPA (dependències) Font: Elaboració pròpia

Per a crear les altres aplicacions independents, els que farem és executar la mateixa comanda i seleccionar *single spa aplication /parcel*. Aquesta tindrà una estructura així:

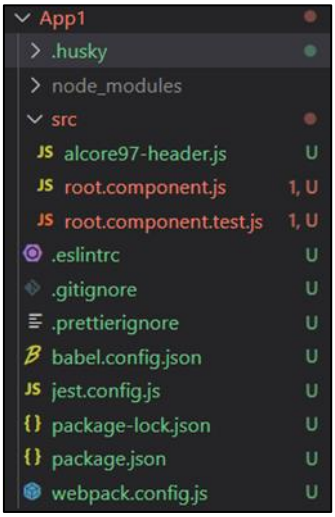


Figura 118: Estrucutra de single-spa aplication/parcel Font: Elaboració pròpia

Root.component.js és el contingut del teu component / MFE.

Alcore97-header.js és el punt d'ancoratge per a poder-lo exportar a l'aplicació root.

Ara només caldrà registrar totes aquestes aplicacions a l'aplicació principal per a poder tenir-les en un mateix entorn!

El resultat final es veuria alguna cosa tan senzilla com això:

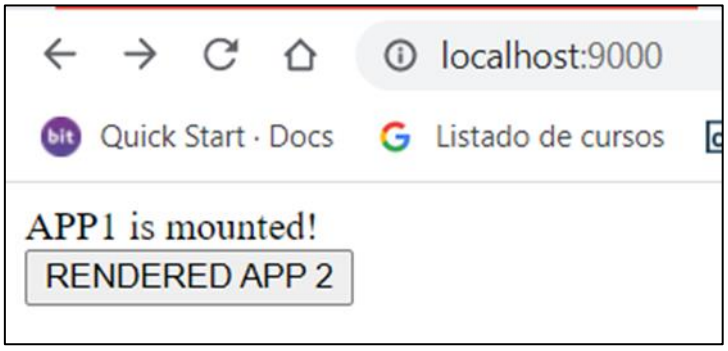


Figura 119: Visualització de dos MFE molt senzills a Single-SPA Font: Elaboració pròpia

Nota: Durant d'instal·lació i prova de concepte d'aquest framework, em vaig trobar amb molts errors d'execució i d'instal·lació. El més notori era que quan tenia l'aplicació muntada volia aixecar la instància em saltàvem diversos errors de codi intern. Per solucionar-ho, vaig afegir a l'arxiu de webpack.config les següents línies:

delete variable.devServer.firewall;

delete variable.devServer.client;

Nota2: És necessari tenir totes les instàncies aixecades per a poder-les visualitzar a l'instància root. A més a més, single SPA te una dificultat afegida, ja que no pots visualitzar les instancies a temps real de manera independent a no ser que facis un seguit de configuracions. Si no les fas, només les pots veure a l'instància root.

8.4 Annex IV: Proba de concepte de Bit

Entrem a veure la última prova de concepte dels frameworks estudiats: Bit, un altre framework molt interessant generador de portals webs que a més a més de capacitat per pujar-ho a la seva pròpia base de dades per l'emmagatzematge d'aquests. Bit, tanmateix també necessita els requeriments de node.js i el ja inclòs gestor de paquets npm o yarn.

En aquest cas, per poder fer ús de bit, hem d'instal·lar, en primera instància, el gestor de versions de bit (BVM):

npm -i -g @teambit/bvm

Un cop instal·lat BVM, hem de procedir a instal·lar, ara sí, BIT:

bvm install

Un cop arribat aquí ja podem crear el nostre espai de treball, bit ens ofereix dues metodologies per a poder crear-lo.

Si volem crear un espai de treball on tots els seus components treballaran sobre React, podem utilitzar la següent comanda:

bit new react <my-workspace-name>

Si volem crear el nostre propi espai de treball i després configurar-lo amb les dependències que nosaltres vulguem, hem d'utilitzar aquesta comanda:

bit init –harmony

Aleshores, un cop inicialitzem el nostre propi entorn de treball, si volem aplicar React a tots els nostres propis components ho fem descomentant aquestes línies:

```
64    "teambit.workspace/variants": {
65      /**
66       * "*" is a special rule which applied on all components in the workspace.
67       **/
68      "*": {
69        /**
70         * uncomment to apply the chosen environment on all components.
71         **/
72         "teambit.react/react": { }
```

Figura 120: Mètode d'inici personalitzat de Bit Font: Elaboració pròpia

I instal·lar les dependències que et convinguin per l'entorn que estem utilitzant.

A continuació en podem a crear components independents o el que serien els nostres MFEs.

Per a fer-ho, podem elegir entre crear un component de mostra JS o TS:

bit create react-component ui/button # TypeScript

bit create react-component-js ui/button # JavaScript

En aquest cas crearem un component que només tindrà un element amb un camp <text>. Tot això estarà dins del fitxer button.tsx o button.js dins de la següent estructura de fitxers:

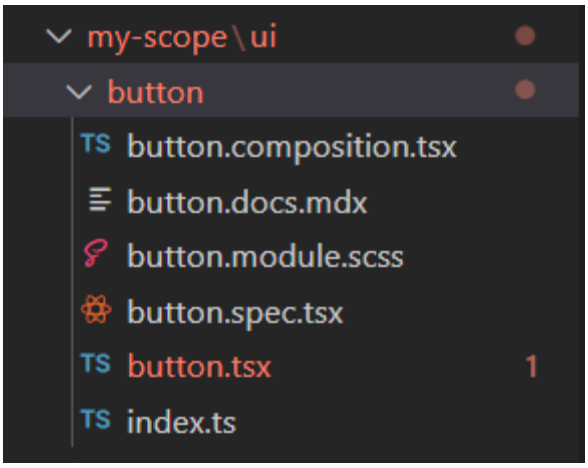


Figura 121: Sistema de fitxers d'un component de bit Font: Elaboració pròpia

Fent una mica de resum de cada arxiu, ho podríem resumir així:

- index.ts → Exporta el button i les *buttonprops*
- button.tsx → Conté el component i les *buttonprops* que li pots passar al component
- button.spec.tsx → Fitxer de testing del component
- button.compositions.tsx → Diferents maneres de tractar un component, podem veure cada tractament reflexat amb un disseny.
- button.docs.mdx --> Documentació

A partir d'aquí nosaltres hem creat un parell de components molt simples: Un botó i una *Card* que conté aquest botó.

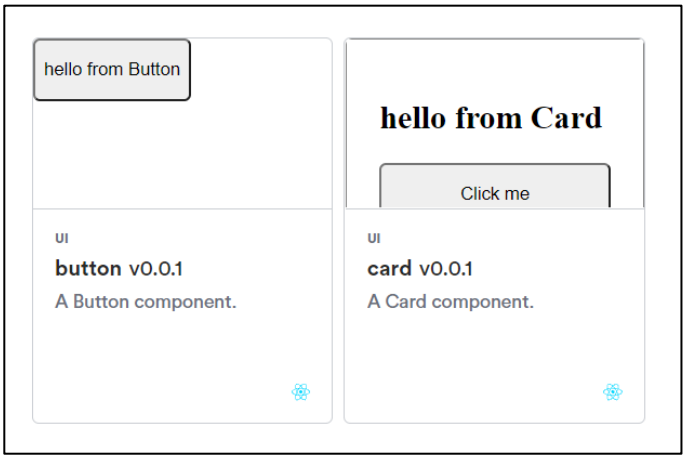


Figura 122: Representació visual dels components creats a Bit Font: Elaboració pròpia

Li hem donat un parell de composicions al botó perquè tinguin un estil diferent, a la pagina ho podem veure així:

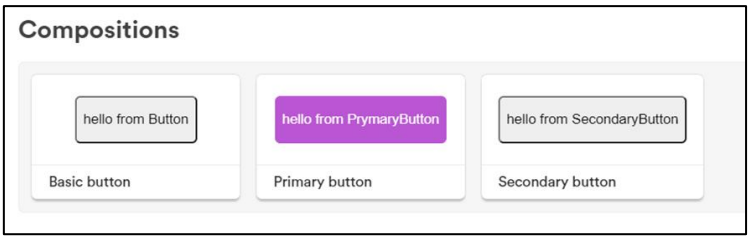


Figura 124: Ús de les composicions a Bit Font: Elaboració pròpia

A més a més que podem veure quins propietats te el component i una petita descripció de que fa cada una:

Figura 123: Propietats de cada components mostrades per Bit Font: Elaboració pròpia

En el cas de la card hem volgut incloure el component del boto prèviament creat. Això a nivell de codi es tradueix a això (quan estem en local):

Figura 125: Referenciació als components de manera local a Bit Font: Elaboració pròpia

Cada cop que fem *bit compile*, compilem el nostre entorn de treballar, posem a punt tots els components creats i "permetem" que altres components puguin importar, dins del mateix entorn, el component en qüestió.

Figura 126: Referencia del nostre entorn de treball al projecte Font: Elaboració pròpia

Però que passa si volem operar tot això com a components independents en entorns independents?

Aquí és on entra Bit.dev, el Cloud de Bit.

Un cop tenim el nostre entorn de treball preparat, ancorem via la CLI el nostre entorn amb el Cloud utilitzant Bit login. Entrem a la nostra sessió a Bit.dev i creem un *Scope*. Aquest *Scope* ens donarà una referència perquè ho podem posar en el nostre arxiu *workspace.jsonc*.

Un cop posat en el fitxer de configuració, hem de connectar els entorns (local i cloud),ho fem amb "Bit link".

Per a pujar els nostres components al Cloud executem "Bit Export". En aquest moment tothom pot importar els components pujats al seu propi entorn de treball. La manera de fer-ho és portant-te el propi entorn del Cloud al teu entorn:

Figura 127: Comanda per importar els components d'una scope Font: Elaboració pròpia

Ara els nostres components els podem utilitzar de la següent manera:

```
3   import { Button, ButtonProps } from '@tfm/workshop.ui.button';
```

Figura 128: Nova manera d'importar els components a Bit Font: Elaboració pròpia

I amb això ja podem crear qualsevol tipus de MFE i treballar àgilment amb Bit!

Nota: Una de les millors característiques que te Bit es l'arbre de dependències de components que es genera automàticament. Veiem a continuació l'exemple anterior amb el boto i la Card:

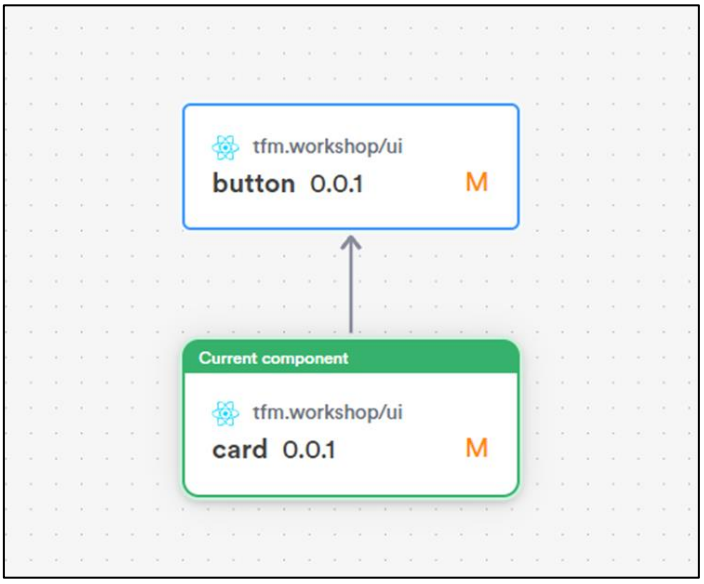


Figura 129: Arbre de dependències de Bit Font: Elaboració pròpia