Alfred Manuel Martínez

Desenvolupament d’una aplicació Android per la plataforma Android Automotive per facilitar el treball dels operaris a l’hora de realitzar assajos als vehicles

TREBALL DE FI DE GRAU

Dirigit per Jordi Massaguer Pla

Grau d'Enginyeria Informàtica

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2024

Resum.

Amb l'avenç de les tecnologies integrades en vehicles, aquest projecte desenvolupa una aplicació per a **Android Automotive**, la qual, facilita les proves d'assaigs en vehicles elèctrics per als operaris responsables de realitzar-les. L'aplicació s'instal·la directament a la pantalla del vehicle equipat amb Android Automotive, permetent als operaris gestionar i agilitzar aquests assaigs de manera més eficaç.

Les proves d'assaig es divideixen en dues categories:

- **Tasks**: Permet als operaris canviar l'estat de les proves mecàniques mentre el vehicle està aturat.
- **Tests**: Facilita la lectura de dades en temps real, com la velocitat i la posició GPS, mentre el vehicle està en moviment.

L'objectiu és optimitzar el procés d'assaigs proporcionant accés ràpid a la informació necessària i automatitzant l'enregistrament dels resultats. Així, es redueix la necessitat de dispositius externs i es minimitza la intervenció addicional d'altres treballadors en el procés.

Resumen.

Con el avance de las tecnologías integradas en vehículos, este proyecto desarrolla una aplicación para **Android Automotive** que facilita las pruebas de ensayos en vehículos eléctricos para los operarios responsables de realizarlas. La aplicación se instala directamente en la pantalla del vehículo equipado con Android Automotive, permitiendo a los operarios gestionar y agilizar las pruebas de manera más eficaz.

Las pruebas se dividen en dos categorías:

- **Tasks**: Permite a los operarios cambiar el estado de las pruebas mecánicas mientras el vehículo está detenido.
- **Tests**: Facilita la lectura de datos en tiempo real, como la velocidad y la posición GPS, mientras el vehículo está en movimiento.

El objetivo es optimizar el proceso de pruebas proporcionando acceso rápido a la información necesaria y automatizando el registro de resultados. De este modo, se reduce la necesidad de dispositivos externos y se minimiza la intervención adicional de otros trabajadores en el proceso.

Abstract.

With the advancement of technologies integrated into vehicles, this project develops an application for **Android Automotive** that facilitates testing on electric vehicles for the responsible operators. The application is installed directly on the screen of vehicles equipped with Android Automotive, allowing operators to manage and streamline the testing process more efficiently.

The tests are divided into two categories:

- **Tasks**: Allows operators to change the status of mechanical tests while the vehicle is stationary.
- **Tests**: Facilitates real-time data reading, such as speed and GPS position, while the vehicle is in motion.

The goal is to optimize the testing process by providing quick access to the necessary information and automating the recording of results. This reduces the need for external devices and minimizes the additional intervention of other workers in the process.

Índex

Índex de taules

Índex de figures

1 Introducció

Aquest projecte presenta el desenvolupament d'una aplicació Android destinada a la plataforma Android Automotive, realitzat en col·laboració amb l'empresa Applus IDIADA.

Applus IDIADA es distingeix com a referent líder en el sector de la consultoria i la prestació de serveis d'enginyeria per a la indústria automobilística. Amb una trajectòria consolidada, l'empresa s'ha distingit per la seva excel·lència en la innovació i el desenvolupament de solucions tecnològiques d'avantguarda per a vehicles.

La missió principal d'Applus IDIADA radica en proporcionar un suport integral als fabricants d'automòbils, des de la concepció del vehicle fins a la seva producció i posada en marxa. Aquest suport abasta una àmplia gamma de serveis, des de proves de vehicle fins a la validació de sistemes avançats d'assistència al conductor i la connectivitat.

Figura 1. Logotip d'Applus IDIADA

En aquest marc d'excel·lència, l'empresa ha decidit encarregar el desenvolupament d'una aplicació innovadora per a la plataforma emergent Android Automotive.

Aquesta iniciativa té com a objectiu principal proporcionar dues funcionalitats als operaris encarregats de fer els assajos, també anomenades *Jobs*, als vehicles elèctrics:

1. Els operaris que duen a terme les proves d'assaig als vehicles en estat parat, també anomenades *Tasks*, se'ls proporcionarà una plantilla de guia de totes les proves d'assaig a realitzar en el vehicle seleccionat, com la verificació de components o el control de sistemes estàtics. Els operaris podran visualitzar i interactuar amb aquestes i disposaran de vària informació rellevant.

   Els mateixos operaris interactuaran des del mateix HMI[1] del vehicle, on es mostraran els camps d'aquestes proves d'assaig a realitzar amb l'objectiu principal de modificar l'estat segons per en quin moment es troba: Defining, Ready, In Progess, In Validation i Done.

2. Els operaris que duen a terme les proves d'assaig en circuit o amb vehicle en moviment, també anomenades *Tests*, se'ls proporcionarà una plantilla de guia de totes les proves d'assaig a realitzar en el vehicle seleccionat, com mesuraments de rendiment o comprovacions dinàmiques.

   L'objectiu és fer una recol·lecció de dades del vehicle en temps real, les dades a recollir són: velocitat, longitud y latitud (GPS), RPM del motor i el nivell de bateria

[1] Human Machine Interface, també coneguda com la pantalla del vehicle

EV[2] durant tot el seu recorregut, les quals s'aniran mostrant per pantalla per donar informació als operaris sobre l'estat del vehicle.

La informació de les proves d'assaig que indiquen als operaris quines Tasks i Tests han de realitzar és gestionada per l'eina TELMA, pròpia de l'empresa.

TELMA és una API REST on estan emmagatzemades i es gestionen totes les proves d'assaig amb la seva respectiva informació, on cada operari tindrà les seves proves assignades a realitzar i consultar els resultats.

L'organització de TELMA és la següent:

1. Els Jobs s'organitzen segons un nom identificat de la prova a operar.
2. Cada Job inclou vàries Tasks i/o Tests a realitzar.
3. Cada Tasks i/o Tests conté camps informatius rellevants per l'operari.

Contextualització de l'aplicació

L'aplicació s'ha dissenyat per funcionar en vehicles que disposen del sistema operatiu Android Automotive, una plataforma desenvolupada per Google específicament per a ser integrada directament en els vehicles. A diferència d'altres sistemes d'infoentreteniment que requereixen un dispositiu mòbil per funcionar, Android Automotive és una versió completa d'Android que actua com el sistema operatiu principal del vehicle. Això vol dir que controla una àmplia gamma de funcions, com la navegació, el control del clima, i altres configuracions del vehicle, tot des de la mateixa pantalla integrada del vehicle.

Android Automotive opera de manera similar a un telèfon intel·ligent, però amb adaptacions específiques per a l'entorn d'un vehicle. Aquesta adaptació proporciona diversos avantatges clau: la integració directa amb els sensors del vehicle, càmeres, i altres sistemes del cotxe permet una interacció més profunda i segura amb el vehicle, oferint informació en temps real que es pot utilitzar per millorar l'experiència de conducció i la seguretat. Per exemple, l'aplicació pot accedir a dades com la velocitat del vehicle, la posició GPS, el nivell de combustible, entre altres, facilitant així la gestió de proves d'assaig per part dels operaris.

Aquesta capacitat d'instal·lar i executar aplicacions directament en el sistema del vehicle fa que Android Automotive sigui una plataforma molt flexible i poderosa per a desenvolupar aplicacions en aquest entorn automobilístic.

Figura 2. Logotip d'Android Automotive

[2] Electric Vehicle

2 Objectius

Optimització del Procés de Proves d'Assaig: A través de l'aplicació, es vol proporcionar als operaris una eina eficaç que els permeti estalviar temps i esforç en la realització de proves d'assaig als vehicles.

Anteriorment, els operaris havien de recórrer a serveis externs per visualitzar les proves a realitzar i de documentar l'estat d'aquestes, però ara tenen la capacitat de visualitzar-les i interactuar directament des del mateix vehicle.

A més, simplifica el procés de registre i lliurament dels resultats de les proves, on ara s'automatitza, eliminant la necessitat d'intermediaris per a aquesta tasca.

Disseny d'una Interfície d'Usuari Adaptativa: També se centra a dissenyar una interfície d'usuari flexible i intuïtiva seguint els límits indicats per Android Automotive per temes de seguretat del conductor.

A més, l'aplicació ha de ser compatible per les diferents pantalles de les diverses marques automobilístiques, on la interfície ha de ser fàcil d'utilitzar per als operaris, independentment del model de vehicle o la seva configuració.

S'aspira a oferir una experiència de navegació fluida i sense errors, permetent una interacció efectiva amb l'aplicació durant el procés de prova d'assaig.

Aplicació Pràctica dels Coneixements Adquirits: L'elecció d'aquest tema per al Treball de Fi de Grau (TFG) es justifica per l'oportunitat d'aplicar els coneixements adquirits durant la carrera en un projecte real amb un impacte tangible com el de desenvolupar una aplicació dirigida a automòbils.

Per finalitzar, la dificultat i el repte tècnic de desenvolupar una aplicació que ha de funcionar en un entorn exigent com el d'un vehicle, el qual s'han de seguir les seves pautes i tenir en compte les seves limitacions a l'hora de programar.

3 Planificació

El projecte s'ha dividit en sis fases principals, cadascuna de les quals defineix un objectiu essencial per al correcte desenvolupament del projecte dins del temps assignat per l'empresa. La planificació assegura que totes les tasques es completin a temps, possibilitant la finalització del projecte amb els riscos degudament mitigats en cada fase.

Diagrama de Gantt

Nom de la tasca	***Duració***	***Comença***	***Acaba***
Fase Inicial projecte	19,47 dies	dj 02/05/24	dc 29/05/24
Fase Disseny	12,25 dies	dc 29/05/24	dv 14/06/24
Fase Programació	39,13 dies	dv 14/06/24	dj 08/08/24
Fase Proves	3,88 dies	dj 08/08/24	dc 14/08/24
Fase Final	4,63 dies	dc 14/08/24	dc 21/08/24
Fase Entrega	6,59 dies	dc 21/08/24	dv 30/08/24

Taula 1. Fases diagrama de Gantt

Fase inicial del projecte

En aquesta primera fase es va decidir que es faria com a projecte per part del Treball de Fi de Grau (TFG). Es van programar reunions setmanals per fer un seguiment detallat del progrés del projecte, es va iniciar la investigació sobre la temàtica i les seves limitacions, i es va procedir a configurar el programari necessari i a entendre l'estructura de la documentació requerida.

Fase Disseny

Aquesta fase és crucial per definir les especificacions del projecte, quins objectius té i els riscos de l'aplicació, quines funcions han de realitzar els operaris perquè compleixin el seu objectiu, com interactuen les dades entre *Front-end* i *Back-end* i quines tecnologies s'utilitzaran per programar.

Els riscos que es van trobar des del principi són: limitacions de programació en vehicles per temes de seguretat del conductor i les limitacions d'aplicació, realitzar els objectius desitjats amb la tecnologia Android Automotive i temes de seguretat d'aplicació que són obligatoris pel compliment de l'empresa.

A més, es van realitzar els dissenys de la UI i la UX de l'aplicació seguint les directrius imposades per Google i Android.

Fase Programació

Aquesta fase marca l'inici del desenvolupament del projecte, en la qual tot el que s'ha definit prèviament es transforma en codi executable. És la fase que requereix més temps, ja que implica convertir les idees en realitat, dissenyar els algorismes necessaris, i adaptar-se a l'entorn de desenvolupament específic.

És important destacar que durant el desenvolupament es van implementar accions obligatòries, com ara el sistema de *Login* empresarial i l'ús de Hardware de seguretat Yubico.

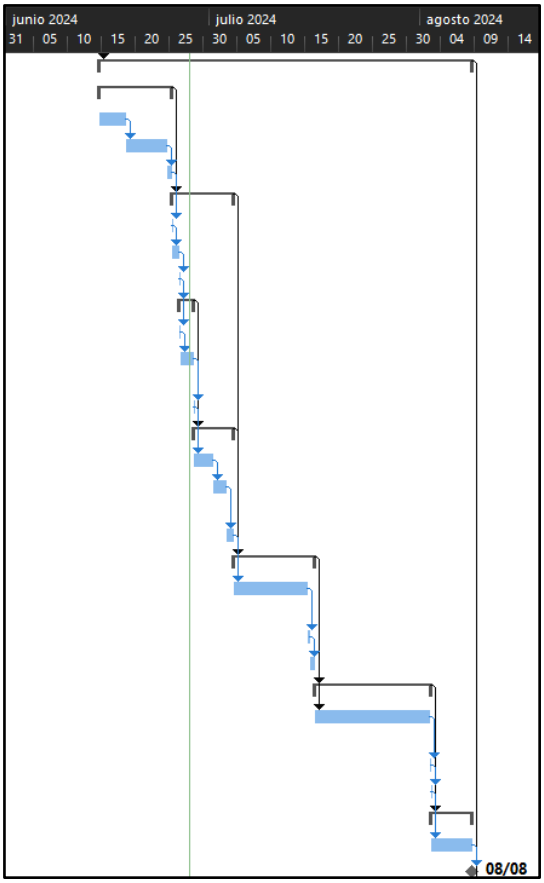

Figura 3. Diagrama Gantt de la Fase Programació

Fase Proves

Aquesta fase és essencial per a la verificació exhaustiva de l'aplicació completada. Es comprova que l'aplicació compleixi amb els objectius establerts i es busca identificar errors de codi, *bugs*, i possibles millores en l'eficiència, estabilitat, i optimització. A més, es preveu provar l'aplicació en un entorn de producció en un vehicle real.

Fase Final

En aquesta fase es finalitza la documentació del projecte i es prepara tot per a l'entrega final.

Fase Entrega[3]

Aquesta és l'última fase del projecte, en la qual es lliura el projecte finalitzat i es realitza la presentació davant el jurat.

[3] Per visualitzar diagrama de Gantt complet: IDIADA/Planificacio.mpp at master · mrallfredii/IDIADA (github.com)

4 Requisits del Projecte

4.1 Diagrama casos d'ús

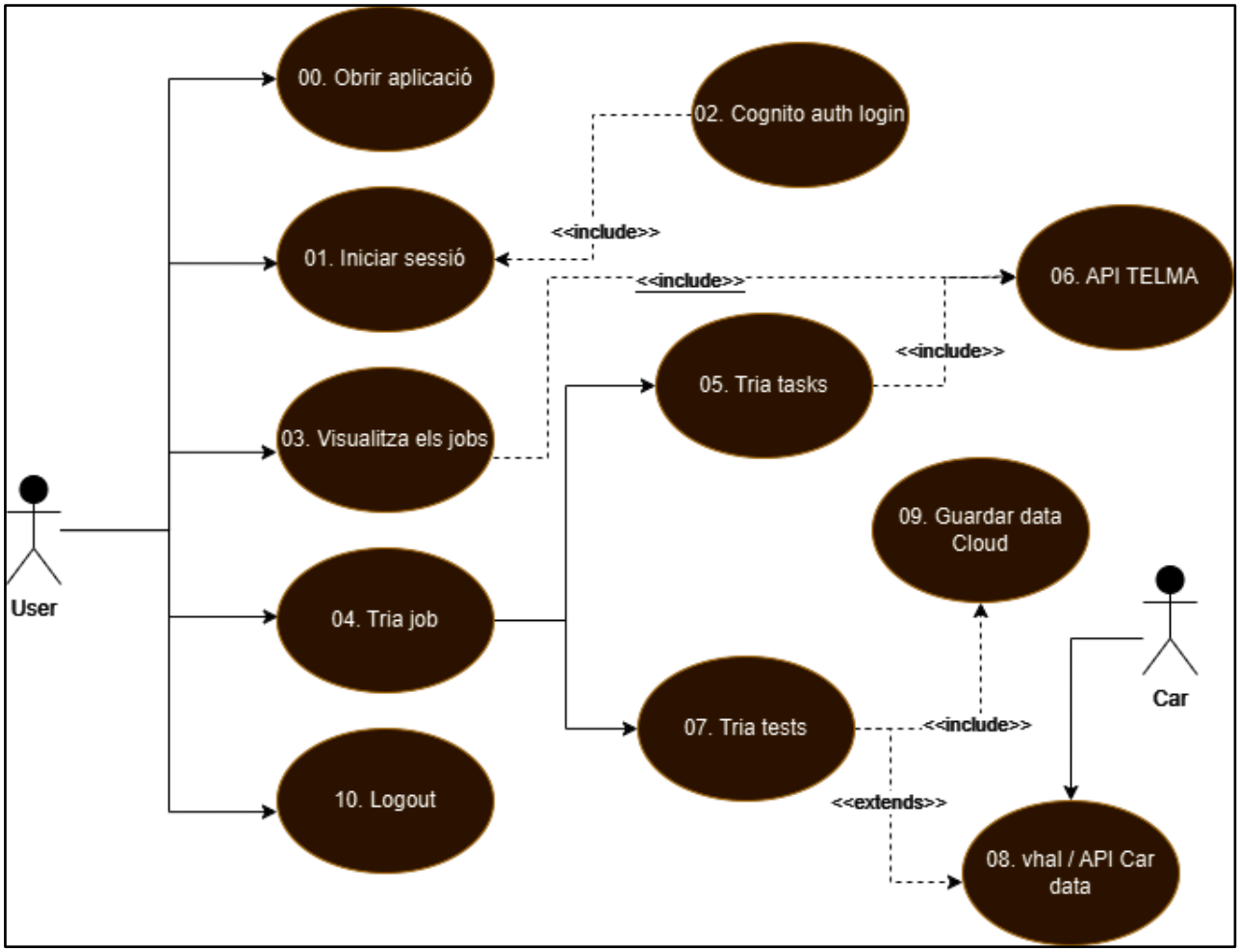

Figura 4. Diagrama casos d'ús

4.2 Requisits Funcionals

4.2.1 00. Guió Obrir aplicació

i. L'usuari ha de poder obrir l'aplicació des de la interfície del sistema Android Automotive
ii. Mostrarà la pantalla d'inici amb informació sobre l'objectiu de l'aplicació
iii. L'operari podrà realitzar l'inici de sessió

4.2.2 01. Guió Iniciar Sessió i 02. Cognito auth

i. Quan l'usuari selecciona l'opció d'iniciar sessió, es redirigeix a una nova pantalla d'explorador web integrada, on haurà d'introduir les seves credencials d'inici de sessió
ii. L'autenticació es realitza mitjançant el sistema Cognito AWS empresarial d'IDIADA, garantint una connexió segura i protegida
iii. Si l'autenticació és exitosa, l'usuari serà redirigit automàticament a la pantalla principal de visualització dels Jobs

4.2.3 03. Guió Visualitzar els Jobs

i. Visualitzar un llistat de Jobs assignats a l'usuari autenticat
ii. L'usuari ha de poder desplaçar-se verticalment pel llistat de Jobs utilitzant la interfície tàctil del vehicle
iii. Incloure l'opció de tancament de sessió (Logout)

4.2.4 04. Guió triar Job

i. L'usuari ha de poder seleccionar un Job del llistat de Jobs
ii. Quan un Job és seleccionat, l'aplicació ha de mostrar una nova pantalla on es visualitzaran totes les Tasks i/o Tests associats al Job
iii. La nova pantalla ha d'incloure una opció per tornar a la pantalla anterior (Visualització de Jobs)

4.2.5 05. Guió Tria Task

i. L'usuari ha de poder seleccionar una Task del llistat de proves associades al Job seleccionat
ii. La pantalla de selecció ha d'incloure una opció per tornar a la pantalla anterior (Triar Job)

4.2.6 06. Guió API TELMA

i. En seleccionar una Task, l'aplicació ha de mostrar una nova pantalla amb els detalls complets de la prova: hora, duració, ordre, etc.
ii. Podrà modificar l'estat de la prova Task: Defining, Ready, In Progess, In Validation i Done
iii. L'usuari ha de poder seleccionar un estat i, posteriorment, guardar els canvis
iv. La pantalla ha de proporcionar l'opció de tornar a la pantalla anterior (Triar Job) o a la pantalla de Visualització de Jobs

4.2.7 07. Guió Tria Test

i. L’usuari ha de poder seleccionar una prova de tipus Test del llistat associat al Job seleccionat
ii. La pantalla de selecció ha d'incloure una opció per tornar a la pantalla anterior (Triar Job)

4.2.8 08. Guió VHAL data

i. L’aplicació ha de gravar les dades del vehicle necessàries per a la prova Test seleccionada, utilitzant la integració amb el Vehicle Hardware Abstraction Layer (VHAL)
ii. Un cop gravades les dades, l’aplicació ha de mostrar una nova pantalla amb els detalls de la prova Test, incloent-hi informació com la velocitat, GPS, RPM, hora, duració, ordre, etc.
iii. L’usuari ha de tenir l’opció de tornar a la pantalla de Triar Job o a la pantalla de Visualització de Jobs en qualsevol moment

4.2.9 10. Guió Logout

i. En qualsevol moment, des de la pantalla de Triar Job, l’usuari ha de poder realitzar el Logout de l’aplicació
ii. En realitzar el Logout, l’aplicació ha de netejar totes les dades de sessió de l’usuari i redirigir-lo a la pantalla d’inici de sessió

4.3 Requisits No Funcionals

4.3.1 Requisit Cicle de Vida

En una aplicació per a Android Automotive, és essencial utilitzar components com Service, Session, i Screen a causa de l'arquitectura específica del sistema operatiu i la manera com aquest interactua amb el maquinari del vehicle i les necessitats d'interacció de l'usuari.

Un **Service** s'ha d'implementar i exportar perquè el sistema host la pugui descobrir i gestionar. És el responsable de validar la confiança en la connexió amb el host. A més, proporciona instàncies de Session per a cada connexió establerta, mitjançant el mètode *onCreateSession*.

Una **Session** representa és una classe abstracta que l'aplicació ha d'implementar i mostrar mitjançant el *CarAppService.onCreateSession*. Aquesta classe actua com el punt d'entrada per presentar informació a la pantalla del vehicle.

Un **Screen** es refereix per gestionar la interfície d'usuari que es mostra a l'usuari final. Aquesta classe té un cicle de vida definit i ofereix els mecanismes necessaris perquè l'aplicació enviï la plantilla que es visualitzarà quan la pantalla estigui activa.

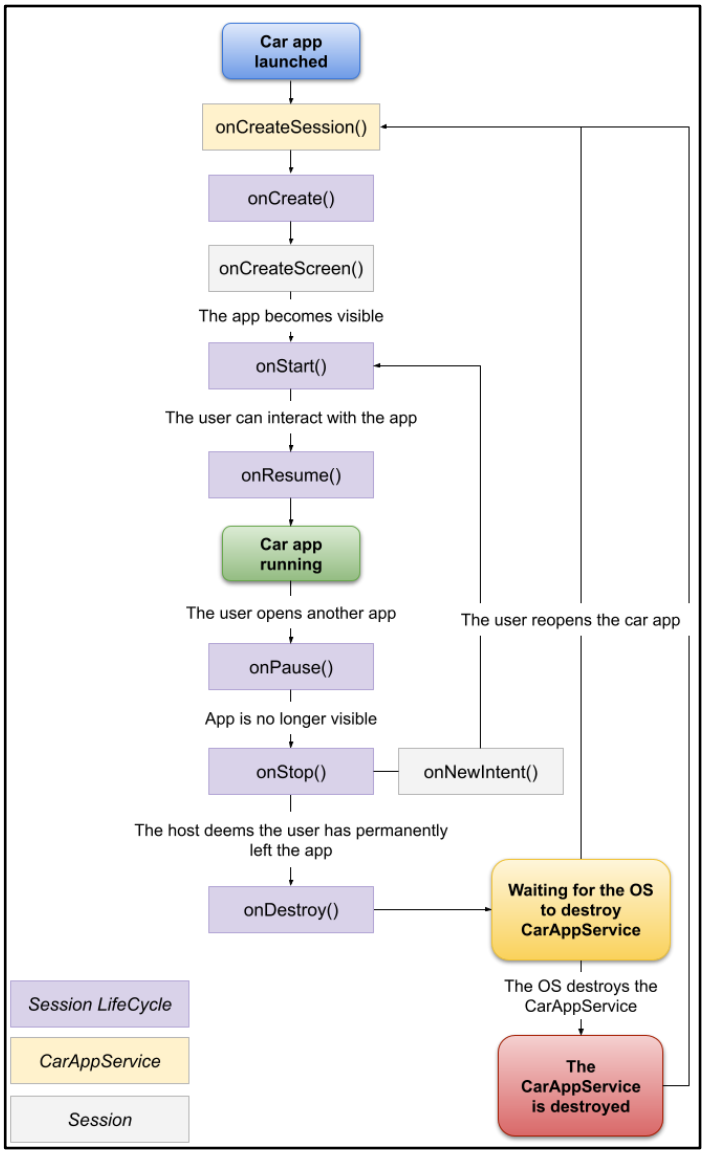

Figura 5. Cicle de vida Android Automotive

4.3.2 Requisit de Rendiment

Per garantir una experiència d'usuari òptima, és crucial que l'aplicació funcioni correctament, sense que es quedi penjada o trigui massa temps a carregar i mostrar la informació.
Les accions de clicar botons, transicions entre pàgines, haurien de ser immediates, en canvi, les accions que impliquen la descàrrega de dades poden variar en temps de resposta segons el volum de dades.

4.3.3 Requisits de Seguretat

L'aplicació ha de garantir que només els usuaris autoritzats puguin accedir a la informació de les proves dels vehicles assignats.

L'aplicació només es podrà executar dins del perímetre de l'empresa, ja que necessita connectivitat Wi-Fi disponible a l'empresa per connectar-se a l'API de TELMA, o bé, des de l'exterior mitjançant una VPN.

Les connexions entre l'aplicació i les crides CRUD es faran utilitzant el protocol HTTP.

Durant l'inici de sessió, els tokens identificatius seran emmagatzemats i encriptats segons els estàndards de Cognito AWS (Bearer) amb una vida útil assignada.

Després de finalitzar les proves, l'operari haurà de fer el Logout, que netejarà totes les dades emmagatzemades per l'aplicació.

4.3.4 Requisits d'usabilitat

L'aplicació ha d'oferir una facilitat d'ús per una bona experiència d'usuari. És a dir, s'ha d'oferir una aplicació amb un disseny simple, intuïtiu, amb missatges d'errors clars.

Per poder provar que l'aplicació es intuïtiva s'haurà de fer amb usuaris reals amb vehicles reals, i per raons organitzatives, l'aplicació haurà de proporcionar l'ús de l'idioma anglès.

4.3.5 Requisis de Manteniment

El codi ha de ser fàcil de mantenir i afegir noves funcionalitats per persones externes al seu desenvolupament. Per aquest motiu el codi ha d'estar ben documentat a més d'utilitzar les bones pràctiques del *framework* i del llenguatge utilitzat.

5 Disseny

5.1 Interfície gràfica i experiència d'usuari

Abans de començar el desenvolupament de l'aplicació, es va decidir dissenyar la interfície d'usuari (UI) i l'experiència d'usuari (UX) seguint les plantilles definides per Android Automotive: **GridTemplate** i **ListTemplate**. Aquestes plantilles són estàndards establerts per garantir una experiència d'usuari consistent i segura dins dels vehicles.

Durant el disseny, es va seleccionar l'HMI del vehicle **Polestar 2**, ja que serà l'emulador que s'utilitzarà pel desenvolupament i test de l'aplicació.

5.2 Dissenys UI i UX

En obrir l'aplicació, es mostrarà una **pantalla de càrrega** amb el logotip de l'empresa. Aquesta pantalla serveix per preparar l'usuari mentre es carrega l'aplicació.

Figura 6. Pantalla de càrrega

Després de la pantalla de càrrega, l'aplicació redirigeix automàticament l'usuari a una **pestanya del navegador web integrada** on es presenta la pàgina d'inici de sessió amb Amazon Cognito. Un cop completat l'inici de sessió, l'usuari és redirigit de nou a l'aplicació principal, minimitzant el nombre de passos necessaris per accedir a les funcionalitats.

Un cop dins l'aplicació, es presenta una pantalla principal amb un llistat vertical de Jobs assignats a l'operari. La plantilla GridTemplate garanteix una organització clara i fàcilment navegable, on cada Job és un element interactiu. L'usuari pot desplaçar-se pel llistat utilitzant gestos tàctils intuïtius, i seleccionar un Job amb un simple toc.

Aquesta pantalla també inclou una opció accessible de Logout, mantenint així el control de la sessió a l'abast de l'usuari en tot moment.

Figura 7. Pantalla per triar Jobs

Un cop seleccionat un Job, l'aplicació mostra una nova pantalla dividida en dues seccions principals, cadascuna representant un tipus de prova: **Tasks** i **Tests**. Segueix un disseny de ListTemplate i divideix clarament les opcions perquè l'usuari pugui identificar ràpidament el tipus de prova que necessita realitzar.

L'usuari pot desplaçar-se pel llistat utilitzant gestos tàctils intuïtius, i seleccionar un Task o un Test amb un simple toc.

Figura 8. Pantalla per seleccionar Tasks o Tests del Job

Si l'usuari selecciona una activitat de tipus Task, es presenta una pantalla dissenyada per mostrar la informació detallada de la prova d'assaig. Aquesta pantalla inclou camps clau com l'hora, la duració, i l'ordre de la prova, tot organitzat en un disseny clar i llegible.

Segueix un disseny de GridTemplate i disposa d'un botó **Change** és un element central i clarament visible que permet a l'usuari modificar l'estat de la prova. Dintre d'aquest botó es disposaran el llistat d'estats de la prova.

Si l'usuari selecciona una activitat de tipus Test, es mostra una pantalla amb la informació rellevant per a les proves en moviment, com la velocitat, la posició GPS, i les RPM del

vehicle. La pantalla està dissenyada per oferir una lectura ràpida i senzilla de les dades, utilitzant el disseny de GridTemplate.

El disseny d'aquesta pantalla té en compte la seguretat del conductor, minimitzant les distraccions mitjançant una interfície neta i optimitzada per a la visibilitat.

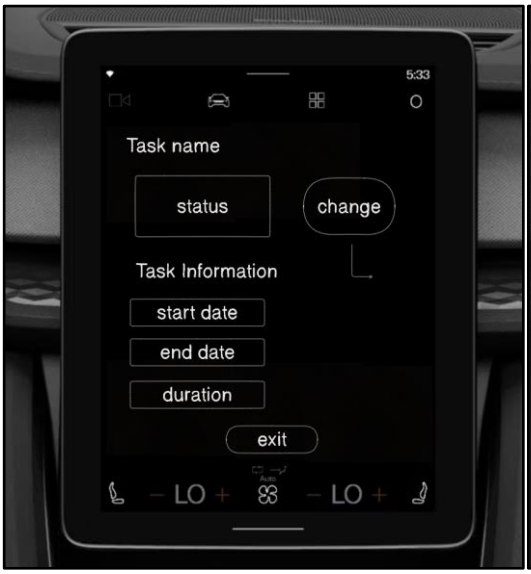

Figura 9. Pantalla de Tasks

Figura 10. Pantalla de Tests

5.3 Flux de connexió Frontend – Backend de les dades empresarials

A continuació es descriuen els fluxos de comunicació entre el *Front-end* i el *Back-end* dins de l'aplicació. Les tecnologies emprades en el Backend són Amazon Cognito i la API REST TELMA.

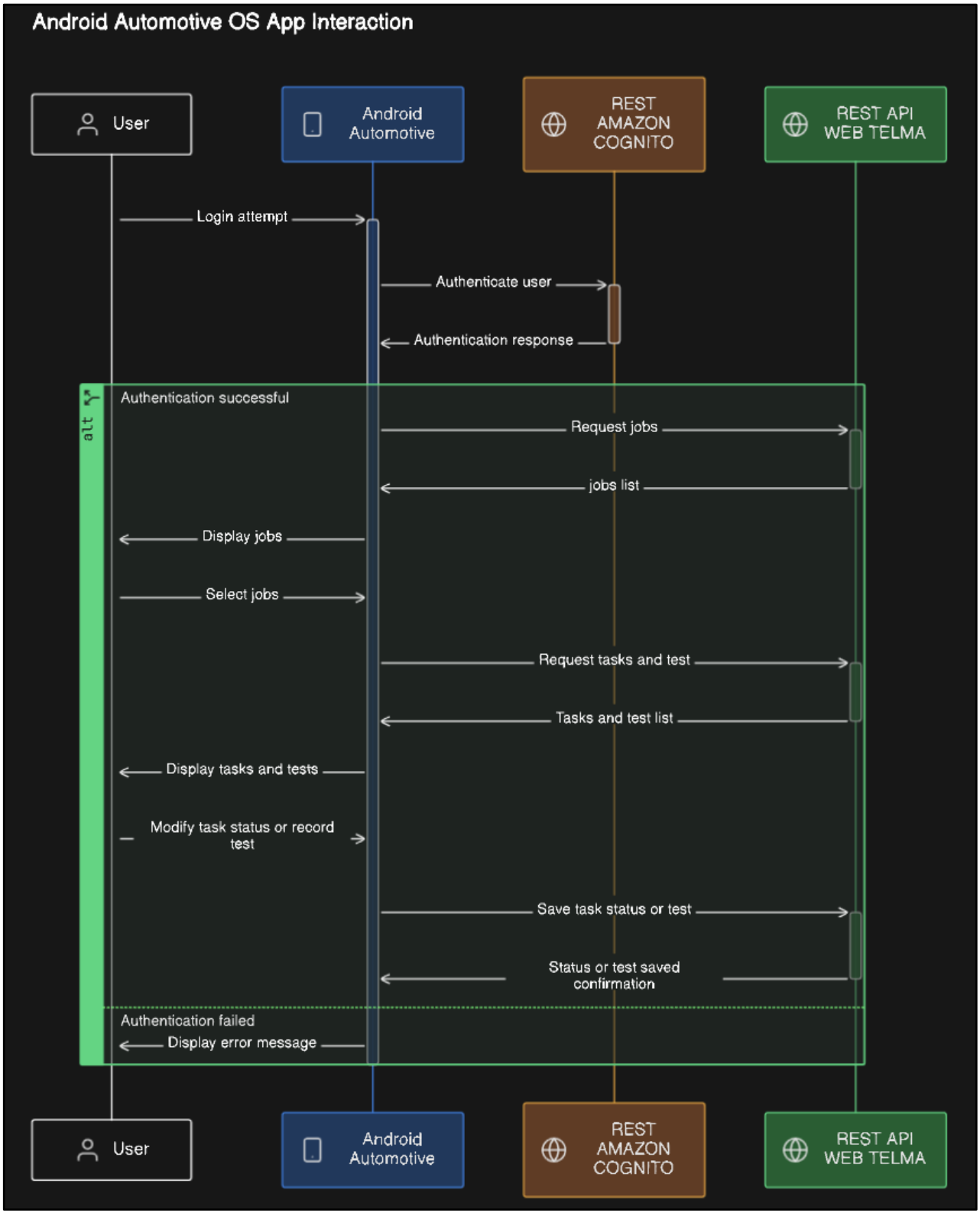

Figura 11. Flux de connexió de dades de l'aplicació

5.4 Arquitectura per obtenir data del vehicle pels Tests

Android Automotive OS (AAOS) és una versió especialitzada d'Android desenvolupada per satisfer els requisits i característiques específiques del sector automobilístic. Tot i compartir el mateix marc de desenvolupament de l'Android Open Source Project (AOSP) que els telèfons intel·ligents, AAOS incorpora components addicionals com el Vehicle Hardware Abstraction Layer (VHAL), el Car Service i la Car API, que són essencials per a la integració amb els vehicles.

El Vehicle Hardware Abstraction Layer (VHAL) actua com una capa d'abstracció que simplifica la complexitat del maquinari d'automoció, convertint la informació procedent dels subsistemes del vehicle en un format normalitzat i comprensible per a les aplicacions. Aquesta abstracció permet que les aplicacions puguin interactuar amb les dades del vehicle sense necessitat de conèixer els detalls específics del maquinari subjacent. La comunicació entre aquesta capa d'abstracció i les aplicacions es realitza a través de la Car API, que s'implementa com un servei del sistema dins d'AAOS. Així, des del punt de vista de l'aplicació, el maquinari del vehicle es gestiona de manera similar a altres perifèrics, com ara dispositius Bluetooth o GPS.

Per tal de comprendre millor el flux de dades entre el maquinari del vehicle i l'aplicació, examinarem amb detall el protocol CAN i les propietats del vehicle, que sovint es fan servir per establir la connexió entre Android i els sistemes dels automòbils.

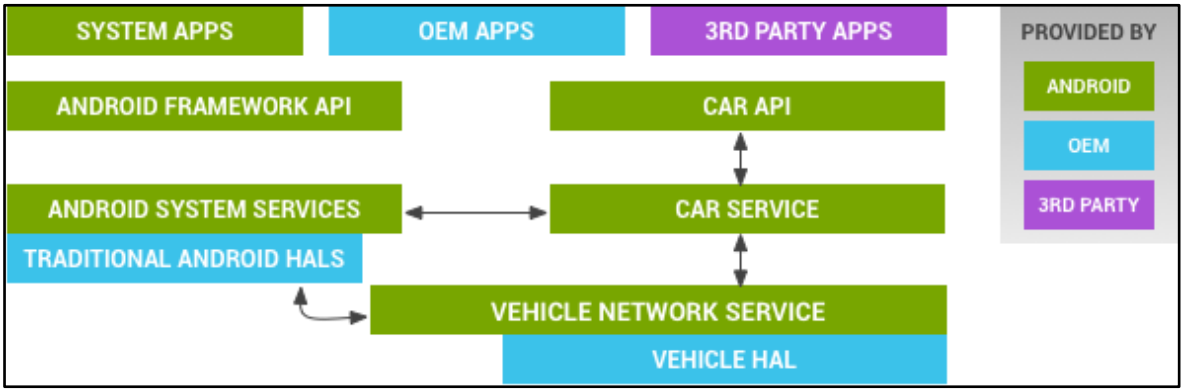

Figura 12. Diagrama arquitectura de vehicles HAL amb Android Automotive

5.4.1 Protocol CAN

Els automòbils moderns estan equipats amb més de 100 unitats de control electrònic (Electronic Control Units, ECU) que es comuniquen entre elles mitjançant diverses xarxes internes del vehicle, com ara el Controller Area Network (CAN), la Local Interconnect Network (LIN) i FlexRay. El sistema operatiu Android Automotive és capaç d'interactuar amb aquestes xarxes, facilitant així la comunicació amb els sistemes del vehicle.

El protocol CAN es fa servir principalment per a l'intercanvi d'informació crítica del vehicle. Aquesta informació inclou dades sobre el moviment del vehicle, com ara la velocitat i l'acceleració, així com l'estat de components essencials com l'accelerador, el sistema de frens, la direcció i la transmissió.

La comunicació CAN es basa en un mètode de transmissió de dades on la diferència de tensió entre dues línies de comunicació es tradueix en bits. El senyal CAN segueix una estructura de trama específica, que inclou diversos elements com el contingut de les dades,

la identificació del node transmissor, l'identificador (ID) que determina la prioritat de l'arbitratge de comunicació, i el camp de dades que conté la informació transmesa, entre d'altres.

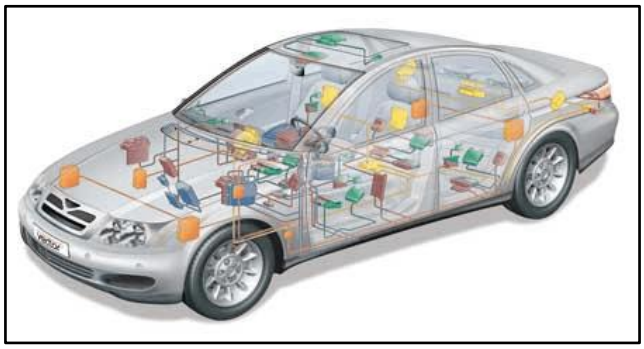

Figura 13. Comunicació CAN

5.4.2 Propietats del vehicle

Android Automotive és capaç de rebre senyals CAN del maquinari del vehicle i proporcionar aquestes dades com a propietats del vehicle en un format que pot ser interpretat per les aplicacions. Aquestes propietats del vehicle es defineixen en un fitxer específic que inclou elements com el Property ID, VehiclePropertyGroup, VehiclePropertyType i VehicleArea, entre altres atributs. La implementació d'aquestes propietats recau en el fabricant d'automòbils, qui és responsable de (1) la definició de les propietats del vehicle i (2) l'algorisme de conversió de senyals CAN a les propietats del vehicle, especificat en fitxers .hal.

Les definicions de les propietats del vehicle es poden trobar al fitxer types.hal, localitzat a *hardware/interfaces/automotive/vehicle/2.0/* dins del codi font de l'AOSP. Aquest fitxer conté la descripció detallada dels principals atributs de les propietats del vehicle, que s'exposen a continuació:

- **Identificador de Propietat**: Serveix com a identificador únic per a cada propietat i es representa en format hexadecimal. Per exemple, en un identificador com 0x0400, 0x indica que es tracta d'un valor hexadecimal, 04 designa la categoria de la propietat, i 00 correspon al nombre dins d'aquesta categoria. En la implementació de referència, les propietats relacionades amb la velocitat del vehicle es classifiquen dins de la categoria 02, com és el cas de PERF_VEHICLE_SPEED amb un identificador de 0x0207.

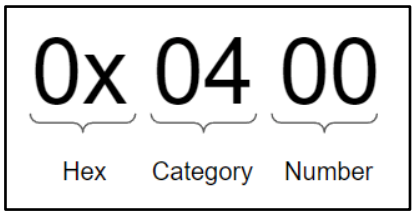

Figura 14. Id de propietat

- **VehiclePropertyGroup**: Aquest atribut identifica si la propietat és una SystemProperty, definida directament dins del sistema operatiu Android Automotive, o una VendorProperty, que conté informació específica del proveïdor o de la implementació particular del vehicle.

- **VehiclePropertyType**: Defineix el tipus de la propietat. Per exemple, si la propietat fa referència a la velocitat del vehicle, el seu tipus serà INT32.

- **VehicleArea**: Aquest atribut indica l'àrea específica del vehicle a la qual es refereix la propietat, com ara WINDOW, MIRROR, etc. El valor resultant és l'OR d'aquests valors, que defineix les propietats concretes del vehicle, per exemple, SENSOR_TYPE_CAR_SPEED, amb un identificador de 291504647 (0x11600207).

5.4.3 Conversió de senyals CAN i propietats del vehicle

El sistema operatiu Android Automotive utilitza una capa d'abstracció de maquinari (CAN HAL) per gestionar la comunicació del bus CAN, que transmet i rep senyals essencials del vehicle. Aquestes senyals són processades pel Vehicle Hardware Abstraction Layer (VHAL) i convertides en propietats del vehicle que poden ser interpretades per les aplicacions.

Aquest procés permet que les aplicacions d'Android Automotive accedeixin a la informació del vehicle en temps real de manera segura i eficient.

El codi font del CAN HAL es pot trobar a *hardware/interfaces/automotive/can/1.0/*.

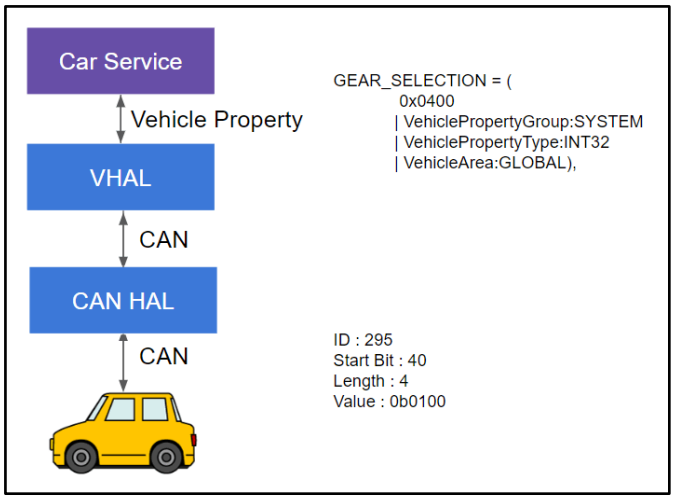

Figura 15. Senyals CAN amb CAN HAL i convertir-los en propietats del vehicle amb VHAL

5.4.4 Interacció entre l'aplicació i la VHAL

La interacció entre l'aplicació i el Vehicle Hardware Abstraction Layer (VHAL) es realitza a través d'una sèrie de serveis i interfícies que faciliten la comunicació amb els subsistemes del vehicle. Les propietats del vehicle, convertides des dels senyals CAN pel VHAL, són rebudes pel Car Service. Aquest servei és responsable de la gestió de la comunicació amb el vehicle dins del sistema Android Automotive.

El Car Service inclou diverses classes clau com CarPropertyService.java, PropertyHalService.java, VehicleHal.java, i HalClient.java, les quals s'encarreguen de passar les propietats del vehicle a les aplicacions. El VHAL exposa mètodes a través d'interfícies que permeten obtenir, establir, i subscriure's als valors de les propietats del vehicle. Aquestes interfícies es poden accedir des de Java com a bytecode, per exemple, a través de IVehicle.hal.class o HalClient.java.

A més, el Car Service interactua amb l'API del vehicle utilitzant AIDL (Android Interface Definition Language), un llenguatge de definició d'interfícies àmpliament utilitzat per a la comunicació entre processos (IPC) a Android. Com que l'aplicació i el Car Service s'executen com a processos separats, es comuniquen mitjançant IPC. Per a la interacció amb les propietats del vehicle, es defineix una interfície específica, i l'API del vehicle proporciona classes que poden ser utilitzades de manera senzilla pel codi Java en el desenvolupament de l'aplicació.

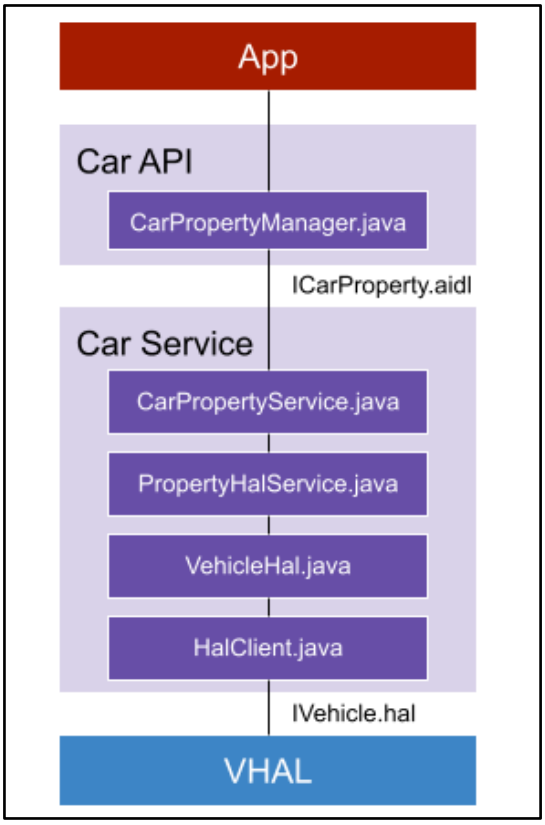

Figura 16. Aplicació amb VHAL

6 Implementació

6.1 Arquitectura Android Automotive

6.1.1 Introducció a l'Arquitectura d'Android Automotive

Android Automotive és una plataforma dissenyada específicament per a vehicles, proporcionant una experiència d'usuari coherent i fluida integrada directament en el sistema d'infoentreteniment del vehicle. Aquesta arquitectura permet que les aplicacions interactuïn de manera segura i eficient amb el maquinari del vehicle, proporcionant funcionalitats adaptades a l'entorn automotriu. En aquest projecte, que s'emmarca dins de l'Automotive IoT, es busca aprofitar aquestes capacitats per desenvolupar aplicacions que interactuïn de manera directa amb els sistemes del vehicle.

6.1.2 Cicle de Vida de l'Aplicació en Android Automotive

Una aplicació en Android Automotive segueix un cicle de vida particular que és essencial per comprendre com es gestiona la connexió i la interacció amb el sistema del vehicle:

Service i Session: Les aplicacions a Android Automotive sovint es basen en serveis per mantenir la comunicació amb altres components del vehicle, com els sensors o altres sistemes d'infoentreteniment. El servei s'inicia quan l'aplicació necessita començar una tasca de llarga durada que pot continuar fins i tot si l'usuari deixa d'interactuar amb l'aplicació. Aquest servei manté una connexió contínua amb el sistema del vehicle i pot gestionar dades en temps real, com en el cas de la recol·lecció de dades del vehicle durant els assajos.

```
public class MyCarAppSession extends Session {
    @NonNull
    @Override
    public Screen onCreateScreen(@NonNull Intent intent) {
        return new HomeScreen(getCarContext());
    }
}
```

```
@NonNull
@Override
public Session onCreateSession() {
    return new MyCarAppSession();
}
```

Screen Lifecycle: Les pantalles d'una aplicació es gestionen dins del cicle de vida de la pantalla, el qual assegura que la interfície només mostri informació rellevant mentre el vehicle està en moviment o aturat, evitant distraccions innecessàries. Les gestions son les transicions de les pantalles segons els canvis d'estat del vehicle. Sempre seguiran el mateix patró:

- **Constructor:** La classe ha d'incloure un constructor que rebi com a paràmetre una instància de *ScreenManager* o una altra classe necessària per a la seva inicialització. Aquesta instància s'utilitzarà per gestionar la pantalla dins del cicle de vida de l'aplicació.

- **onGetTemplate:** El mètode onGetTemplate() defineix quin tipus de plantilla s'utilitzarà per renderitzar la interfície d'usuari d'aquesta pantalla.

  @NonNull

  @Override

  public Template onGetTemplate() { // Implementació del codi per obtenir la screen }

A més, es necessita la classe *CarContext*, que és una subclasse de *ContextWrapper* accessible des de les classes Session i Screen. CarContext proporciona accés a serveis específics del vehicle, com ara el *ScreenManager* per gestionar la pila de pantalles, i el *AppManager* per accedir a dades generals relacionades amb la funcionalitat de l'aplicació.

6.1.3 Plantilles utilitzats en Android Automotive

L'arquitectura d'Android Automotive proporciona una sèrie de plantilles (templates) que s'utilitzen per dissenyar interfícies d'usuari segures i coherents. Cadascuna té els seus requisits UX. Les utilitzades en aquest projecte són:

Grid Template: Aquest és ideal per a mostrar elements en una disposició de graella (grid), proporcionant una visió clara i accessible de múltiples opcions o informacions alhora, com els Jobs.

List Template: El template de llista és més adequat per a mostrar elements en una forma seqüencial, com una llista de tasques o tests a realitzar pels operaris.

Message Template: Aquest s'utilitza per mostrar missatges importants a l'usuari, com alertes o notificacions que requereixen una acció immediata. En el meu context, això és crucial per a alertes de seguretat en l'API.

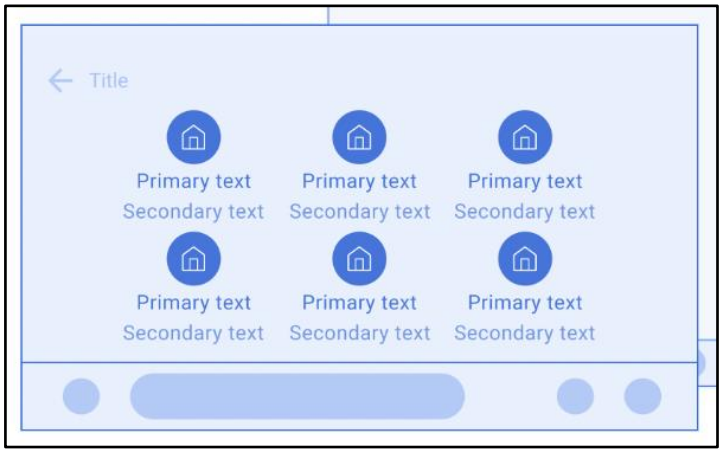

Figura 17. Disseny UI de GridTemplate

6.1.4 Programació i configuració en Android Automotive

La programació en Android Automotive comparteix moltes similituds amb el desenvolupament d'aplicacions Android convencionals, però amb algunes diferències clau per adaptar-se a l'entorn d'un vehicle. Aquest apartat tracta sobre com es programa efectivament per Android Automotive, incloent-hi com es manegen les transicions de

pantalla, la interacció amb l'usuari, i l'ús de components específics de la plataforma. Per configurar un projecte Android Automotive aneu a l'apartat configuració d'Annexos.

A Android Automotive, la gestió de les pantalles no es fa directament amb *Activities* com en les aplicacions Android estàndard. En canvi, es fa ús del ScreenManager, que és una part integral del sistema per gestionar les transicions de pantalla dins de la interfície d'usuari del vehicle. El ScreenManager permet controlar com es presenten les pantalles dins de l'aplicació i facilita la navegació entre elles d'una manera que minimitza les distraccions.

El fitxer AndroidManifest.xml és clau per definir les característiques, permisos, activitats, i serveis que l'aplicació utilitza, especialment quan es desenvolupa per a Android Automotive. Aquest fitxer inclou la declaració de característiques com *android.hardware.type.automotive*, que assegura la compatibilitat amb dispositius automotrius, indicant que l'aplicació només es podrà instal·lar en vehicles equipats amb aquest sistema operatiu.

L'aplicació està dissenyada exclusivament per a dispositius amb *Hardware* de tipus automotriu:

```
<uses-feature
   android:name="android.hardware.type.automotive"
   android:required="true" />
```

Aquesta secció defineix un servei essencial per a la interacció de l'aplicació amb el sistema del vehicle (*MyCarAppService* gestiona les comunicacions amb el sistema AAOS):

```
<service
   android:name=".MyCarAppService"
   android:exported="true">
   <intent-filter>
      <action android:name="androidx.car.app.CarAppService"/>
      <category android:name="androidx.car.app.category.IOT"/>
   </intent-filter>
</service>
```

El fitxer *automotive_app_desc.xml*, conté informació detallada sobre com l'aplicació ha de comportar-se dins del vehicle:

```
<meta-data android:name="com.android.automotive"
   android:resource="@xml/automotive_app_desc"/>
```

6.2 Iniciar sessió amb Amazon Cognito Amplify

El procés d'inici de sessió de l'aplicació es basa en les tecnologies Amazon Cognito i Amplify, proporcionades per Amazon Web Services (AWS). Aquestes tecnologies gestionen l'autenticació i autorització dels usuaris mitjançant una pàgina web externa pròpia d'IDIADA, integrada amb AWS, on es realitza el Login empresarial.

Amazon Cognito s'utilitza per configurar un *user pool*, un directori d'usuaris per a l'autenticació i autorització de les aplicacions Android. Des de la perspectiva de l'aplicació, un *user pool* d'Amazon Cognito actua com a proveïdor d'identitat (IdP) compatible amb OpenID Connect (OIDC).

La integració amb Cognito es realitza a través *d'endpoints* a la seva API, on es passen diversos camps a l'URI per identificar el *user pool* i gestionar el procés d'autenticació. Aquests camps són essencials per identificar els usuaris, els *user pools*, el lloc de redirecció, entre altres paràmetres.

Endpoint GET /login: la crida invocarà una interfície d'usuari d'inici de sessió allotjada en una web pròpia mitjançant un navegador, on demanarà iniciar sessió amb les credencials empresarials. La resposta envia un codi identificador i únic d'usuari.

GET https://mydomain.auth.regio.amazoncognito.com/login?

response_type=code&

client_id=nom_id_clients&

redirect_uri=https://YOUR_APP/redirect_uri&

state=STATE&

scope=openid+profile+aws.cognito.signin.user.admin

Els camps a introduir en la crida van ser proporcionats per l'empresa, i l'URI de redirecció es va assignar com 'myapp/callback'.

Endpoint POST oauth2/token: amb el codi obtingut a la resposta, podrem obtenir diversos tokens: token d'accés, el qual és necessari per obtenir la metadada de l'usuari, token identificador, que identifica l'usuari autenticat i el token d'actualització, el qual dona un temps de vida d'aquest.

Endpoint GET oauth2/userInfo: a partir del token d'accés anterior es podrà fer una crida per obtenir un JSON de retorn amb els camps d'informació de l'usuari que ha iniciat sessió correctament, en el meu cas agafaré el nom de l'usuari.

Endpoint GET /logout: aquesta crida far tancar la sessió de l'usuari i redirigeix a un URL de tancament de sessió autoritzat per al client de l'aplicació. A més, es netegen les dades emmagatzemades per l'aplicació per garantir la seguretat.

Per realitzar aquestes crides REST, s'ha utilitzat HttpURLConnection per gestionar les connexions HTTP, verificant sempre si les crides han estat exitoses amb *OnSuccess* o si han fallat amb *OnError*.

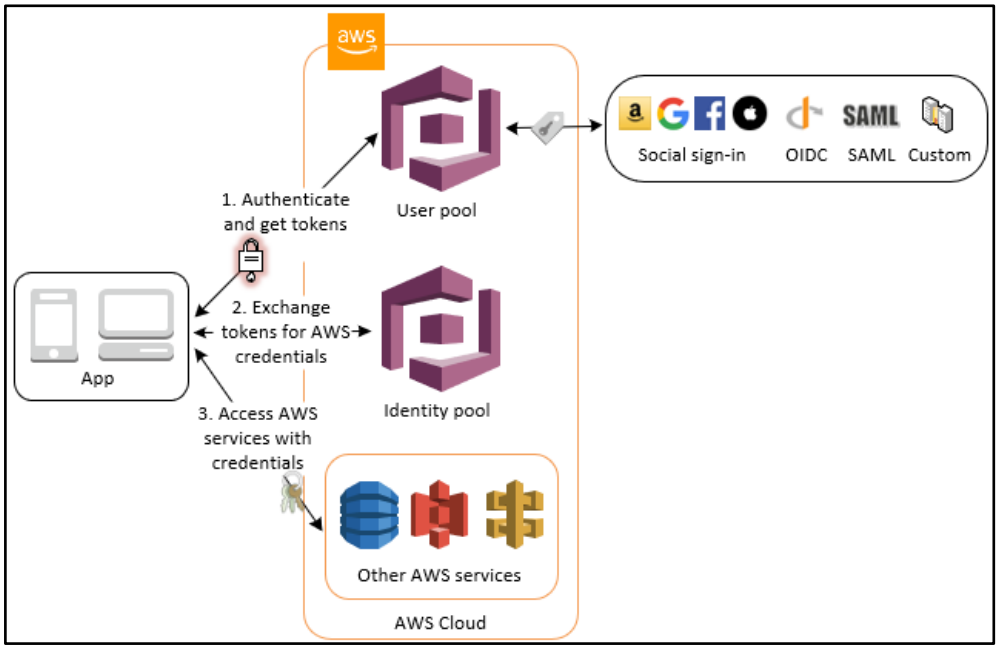

Figura 18. Arquitectura AWS Cognito inici sessió

6.3 API REST TELMA

Els operaris tenen assignats els seus Jobs a través de TELMA, una eina desenvolupada per APPLUS IDIADA que emmagatzema les dades de les proves que s'han de realitzar als vehicles.

TELMA registra informació com la duració de les proves, la data de realització, l'estat, les tasques a realitzar sobre el vehicle, els usuaris responsables, i altres condicions rellevants per assegurar el compliment dels objectius establerts.

Aquesta eina es gestiona com una API REST, permetent l'emmagatzematge i la modificació de les condicions de les proves, així com l'actualització dels resultats finals.

Per a la implementació de les crides REST a l'API de TELMA, s'ha utilitzat Retrofit2, una llibreria Java per a Android que simplifica la recuperació i càrrega de dades JSON mitjançant REST. Retrofit2 s'ha triat per la seva facilitat d'ús, la seva llegibilitat, i el seu suport per a la conversió automàtica de dades JSON a objectes Java mitjançant *GsonConverterFactory*.

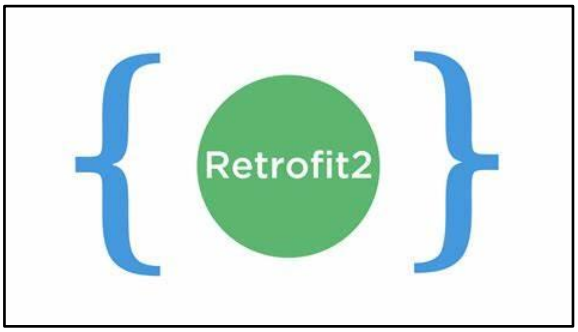

Figura 19. Logotip Retrofit2

Endpoint GET /jobs

Aquest endpoint es crida immediatament després de l'autenticació de l'usuari, i retorna una llista de tots els Jobs assignats.

La crida retorna un llistat de tots Jobs, on cada Job conté informació rellevant, incloent-hi el nom i la seva ID, que són crucials per a la següent crida. La informació es passa a la següent pantalla a través del constructor.

```
.setOnClickListener(() -> getScreenManager().
push(new JobDetailScreen(getCarContext(), job.getId(), job.getName())
))
```

Endpoint GET /jobs/{jobsId}

Quan l'usuari selecciona un Job, es crida aquest endpoint amb la ID del Job seleccionat. La resposta és un JSON que conté les dades específiques de la prova corresponent. Les dades es passen a la següent pantalla mitjançant un *Bundle* per a Tasks o a través del constructor amb *ScreenManager* per a Tests.

Mètode per agafar la data d'un Job seleccionat:

```
  private void fetchJobDetails() {
      String idToken = SharedPreferencesUtil.getIdToken(getCarContext());
      Telma telma = new Telma();
      telma.getJobDetails(idToken, jobId, new
Telma.TelmaCallback<JobDetailResponse>() {

          @Override
          public void onSuccess(JobDetailResponse jobDetailResponse) {
              jobDetail = jobDetailResponse.getData();
              // Refresh the screen to display the job details
              invalidate();
          }
          @Override
          public void onError(Exception e) {
              Log.d("Screen Jobs info", String.valueOf(e));
          }
      });
  }
```

Endpoint PUT /jobs

Aquest *endpoint* es crida quan l'operari fa canvis en l'estat d'una Task del vehicle, modificant i emmagatzemant el nou estat de la prova a l'API TELMA.

Mètode per retornar el *body* de la resposta de la crida:

```
public void updateJobStatus(String idToken, DataItem execution, TelmaCallback<Void> callback) {
    Call<Void> call = apiService.updateJob("Bearer " + idToken, execution);
    call.enqueue(new Callback<Void>() {

        @Override
        public void onResponse(@NonNull Call<Void> call, @NonNull Response<Void> response) {
            callback.onSuccess(response.body());
        }

        @Override
        public void onFailure(@NonNull Call<Void> call, @NonNull Throwable t) {
            callback.onError(new Exception(t));
        }
    });
}
```

Ús de POJOs:

Les classes POJO (Plain Old Java Object) s'utilitzen per deserialitzar les respostes JSON. Aquestes classes encapsulen la lògica de negoci i s'utilitzen per convertir el format JSON en objectes Java, facilitant la manipulació de les dades retornades o enviades. A causa de la complexitat de les dades emmagatzemades, aquestes classes es generen mitjançant *plugins*.

Definició de la interfície API:

Codi per declarar les interfícies de l'API, on segueixen sempre la següent estructura:

L'anotació @GET especifica el mètode de sol·licitud en aquest cas estem fent una sol·licitud GET a aquest URL.

L'anotació @Header, @Path i @Body especifiquen paràmetres de consulta.

@GET("/job")

Call<ApiResponse> getJobs(@Header("Authorization") String idToken);

@GET("/job/{jobId}")

Call<JobDetailResponse> getJobDetails(@Header("Authorization") String idToken, @Path("jobId") String jobId);

@PUT("/job")

Call<Void> updateJob(@Header("Authorization") String idToken, @Body DataItem job);

Classes per definir el model POJO per les dades de retorn i entrega:

Codi per definir les dades del format de JSON que rebem o enviem, per fer els *setters* i *getters* i poder obtenir del retorn així les dades que necessitem, definits en el paquet data.

private String id;

public String getId() { return id; }

public void setId(String id) { this.id = id; }

Classe per fer les crides a la interfície API:

Codi per utilitzar les interfícies API, sempre segueix la següent estructura:

Retrofit retrofit = new Retrofit.Builder()

.baseUrl(ENDPOINT)

.addConverterFactory(GsonConverterFactory.create())

.build();

6.4 API Car

Com s'ha mencionat anteriorment, per obtenir les dades del vehicle en un entorn Android Automotive, utilitzarem l'API Car. Ara en centrarem en la part que converteix el senyal CAN en les propietats de la carrosseria del vehicle. Això s'implementa a *VHAL*, on es requereix:

6.4.1 Car Managers

CarPropertyManager: Aquest *manager* permet accedir a diverses propietats del vehicle, incloent-hi la velocitat, les revolucions per minut (RPM) del motor, i el nivell de bateria. Aquesta interfície facilita la lectura de propietats específiques del vehicle i la recepció d'actualitzacions en temps real.

VehiclePropertyIds: Conté constants que representen diferents propietats del vehicle, com la velocitat del vehicle, les RPM del motor, i el nivell de bateria, entre d'altres.

Maneig de Propietats del Vehicle:

- registerVehicleSpeedListener(): Registra un *callback* per escoltar canvis en la velocitat del vehicle (VehiclePropertyIds.PERF_VEHICLE_SPEED).

- registerEngineRpmListener(): Registra un *callback* per rebre actualitzacions de les RPM del motor (VehiclePropertyIds.ENGINE_RPM). Aquest listener només es registra si hasCarEnginePermission és *true*.

- registerBatteryLevelListener(): Registra un *callback* per monitorar el nivell de la bateria del vehicle elèctric (VehiclePropertyIds.EV_BATTERY_LEVEL). Aquest listener només es registra si hasCarEnergyPermission és *true*.

El callback *carPropertyManager.registerCallback* es cridarà a la devolució de trucada registrada quan hi hagi un canvi de valor.

I on, per exemple, PERF_VEHICLE_SPEED obté la velocitat del vehicle gràcies a l'explicat a l'apartat **5.4 Arquitectura per obtenir data del vehicle pels Tests**.

carHelper

```
carPropertyManager.registerCallback( new
CarPropertyManager.CarPropertyEventCallback() {
  @Override
  public void onChangeEvent(CarPropertyValue value) {
    if (value.getPropertyId() == VehiclePropertyIds.PERF_VEHICLE_SPEED) {
       currentVehicleSpeed = (Float) value.getValue();
       if (onCarDataChangeListener != null) {
onCarDataChangeListener.onVehicleSpeedChange(currentVehicleSpeed); }
    });
  }
}
```

6.4.2 Seguretat

Per accedir a les dades del vehicle, els usuaris hauran de concedir permisos específics a l'aplicació. Aquestes dades tenen diferents nivells de protecció, classificats com a permisos normals, perillosos, o privilegiats.

El codi està dissenyat per gestionar situacions en què l'aplicació pot no tenir accés a certes dades del vehicle. Depenent dels valors retornats per les funcions hasCarEnginePermission i hasCarEnergyPermission, els *listeners* per a les propietats corresponents es registraran o no. Això és crucial, ja que l'accés a dades confidencials del vehicle requereix permisos específics que han d'estar declarats en l'arxiu AndroidManifest.xml.

TestScreen

```
getCarContext().requestPermissions(
  Arrays.asList("android.car.permission.CAR_SPEED),
            (grantedPermissions, rejectedPermissions) -> {
               if (grantedPermissions.contains("android.car.permission.CAR_SPEED") {
                  carHelper = new CarHelper(getCarContext(), this,
grantedPermissions.contains("android.car.permission.CAR_ENGINE_DETAILED"),
grantedPermissions.contains("android.car.permission.CAR_ENERGY"));
                  initializeLocationServices();
               }
            });
```

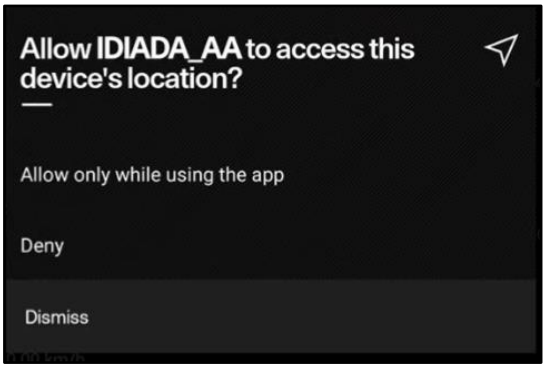

Figura 20. App demana permisos a l'usuari de localització

6.4.3 AndroidManifiest.xml

En el manifest s'han d'introduir les variables necessàries per llegir les dades del vehicle, són essencials per determinar quines dades i funcionalitats del vehicle poden accedir a l'aplicació.

```
<uses-permission android:name="android.car.permission.CAR_INFO"/>
<uses-permission android:name="android.car.permission.CAR_SPEED"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.car.permission.CAR_ENGINE_DETAILED" />
<uses-permission android:name="android.car.permission.CAR_ENERGY"/>
```

7 Avaluació

7.1 Joc de Proves Polestar 2

En aquesta secció es realitza un joc de proves complet per simular el procés que seguiria un operari de l'empresa durant l'execució d'una Task i un Test en un vehicle Polestar 2. Per a aquesta avaluació, s'ha creat un Job fictici anomenat ALFRED_JOB_1, que representa un Job real de l'empresa, amb l'objectiu de poder mostrar el funcionament de l'aplicació sense exposar dades sensibles.

7.1.1 Task

PAS 1. Obrir l'aplicació al vehicle, es mostrarà la pantalla principal per fer l'inici de sessió. Aquí mostra informació sobre l'aplicació i mostra el botó per al Login.

PAS 2. Si cliquem el botó Login, es mostra la pantalla del Login Cognito d'Applus IDIADA mitjançant el navegador Vivaldi, aquí l'operari introdueix el correu i la contrasenya.

Figura 21. Pas 1

Figura 22. Pas 2

PAS 3. Un cop validat l'inici de sessió, a l'aplicació es mostrarà la pantalla amb la crida REST a Jobs mitjançant la GridTemplate incloent el nom de l'operari i el botó del Logout per sortir.

PAS 4. Se selecciona el Job ALFRED_JOB_1, i es mostrarà la següent pantalla amb la informació de Task i Test assignats mitjançant una ListTemplate. En aquest cas els noms identificatius també es van assignar inventats per fer el joc de proves.

Figura 23. Pas 3

Figura 24. Pas 4

PAS 5. Ara voldrem fer un canvi d'estat de la Task: Task oscar.

Per això clicarem sobre d'ella i mostrarà la següent pantalla amb la informació de la Task mitjançant una GridTemplate.

PAS 6. Obtenim la informació de la prova a realitzar, podem veure que hi ha camps que ens donen informació rellevant.

Ara aquesta Task està amb status Definig, i la volem passar a Done, perquè suposem que ja hem acabat aquest Task. Per fer això haurem de clicar a Update status, on es mostrarà una nova pantalla amb tots els tipus d'estat mitjançant una ListTemplate.

Figura 25. Pas 5

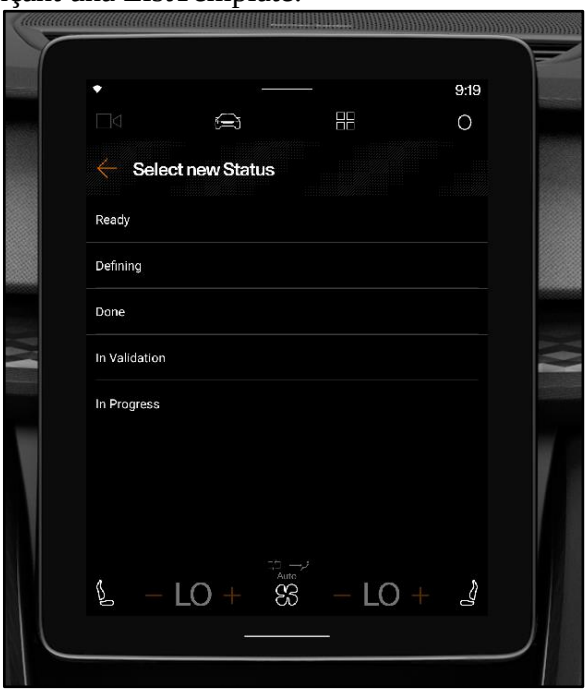

Figura 26. Pas 6

PAS 7. Seleccionem l'estat Done, es tancarà aquesta pantalla automàticament, i es produirà el canvi de pantalla al nou estat. Aquest canvi fa la crida REST @PUT a TELMA per fer el *Update* de l'estat.

També s'actualitzarà el camp *Modified by* segons quin operari ha fet els canvis d'estat.

Podrem observar el canvi:

Figura 27. Pas 7

I si volem comprovar el canvi d'estat a l'eina de TELMA, podem accedir a la pàgina web de Telma, accedir al Job i observar el camp *Status*.

Order	Type [Task/Test]	Activity Name	Duration [h]	Acronym / Type	Start Date	End Date	Status	Actions
	Select one						Select one	
1	Task	Task oscar	1	task_oscar	20/Jun/2024	20/Jun/2024	job.in validation	
2	Test	Test oscar	1	test_oscar	20/Jun/2024	20/Jun/2024	In progress	
3	Task	task	3	ttt	20/Jun/2024	20/Jun/2024	job.defining	
4	Test	Test oscar	1	test_oscar	20/Jun/2024	20/Jun/2024	Defining	

Figura 28. Canvis d'estat en l'eina de Telma

7.1.2 Test

PAS 8. Un cop finalitzada aquest Task, voldrem fer una prova de Test.

Voldrem fer el Test: Test Oscar, per anar a aquesta pantalla, podrem seleccionar el botó Task done.

Es mostrarà la pantalla de Job amb Task i Test, i seleccionem la desitjada, a continuació, anirem a la pantalla del Test, on es mostrarà tota la informació rellevant d'aquesta.

Figura 29. Pantalla Tasks i Tests

Figura 30. Pas 8

PAS 9. Ara voldrem fer una prova per poder veure els canvis de velocitat del vehicle, i poder observar en temps real per pantalla la velocitat actual d'aquest.

Per això disposem de Polestar – Extended Controls, el qual és una centraleta on es pot anar modificant els valors de velocitat, marxa (gear), fre de mà, etc. A més de disposar de sensors del vehicle a la pestanya de VHAL *properties*, els quals, també es poden llegir en temps real.

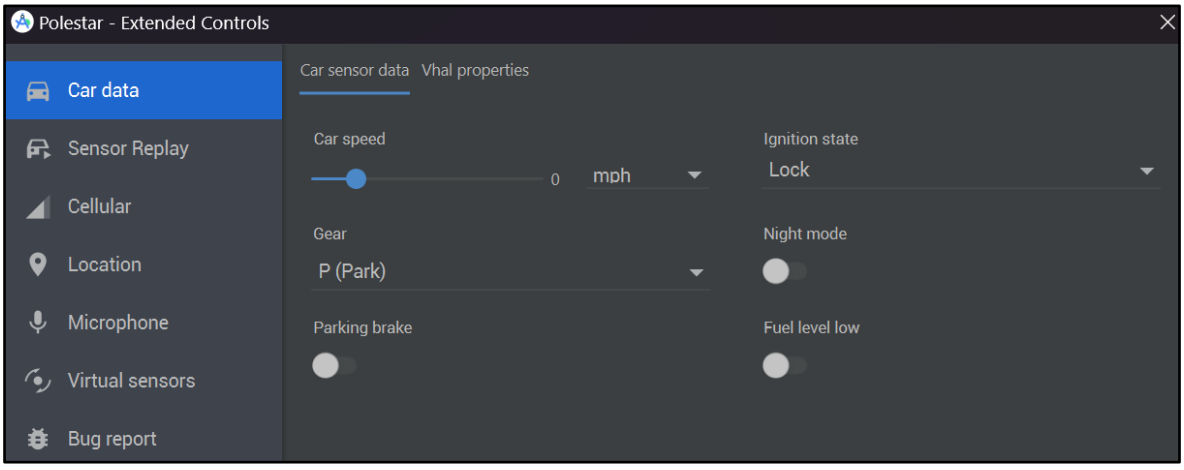

Figura 31. Pas 9, Polestar – Extended Controls

En aquest cas, jugarem amb l'apartat de Car speed, el qual l'anirem augmentant amb el ratolí cap a la dreta, fent així, augmentat i accelerant la velocitat del vehicle.

Els canvis s'aniran reflectint per la pantalla en temps real.

En un vehicle real, aquest apartat seria el mateix vehicle en si, i Car speed seria el mateix pedal d'accelerador.

A més, també es podrà observar altres camps, com el nivell de bateria EV del vehicle per pantalla.

Figura 32. Test velocitat 1

Figura 33. Test velocitat 2

7.2 Joc de Proves en altres models de vehicles

A més de les proves realitzades en el vehicle Polestar 2, l'aplicació s'ha testejat en altres models de vehicles i OEMs per garantir la seva compatibilitat i funcionalitat en diferents entorns.

7.2.1 Honda

En aquesta prova, es va utilitzar un model genèric d'Honda, proporcionat per la pròpia empresa Honda com un emulador amb Android Automotive. Aquest emulador ofereix una experiència visual similar a la dels vehicles reals de la marca. Tot i ser una marca diferent, es poden observar els dissenys i interfícies específiques del fabricant (OEM), així com la integració de l'aplicació en l'entorn gràfic particular d'Honda.

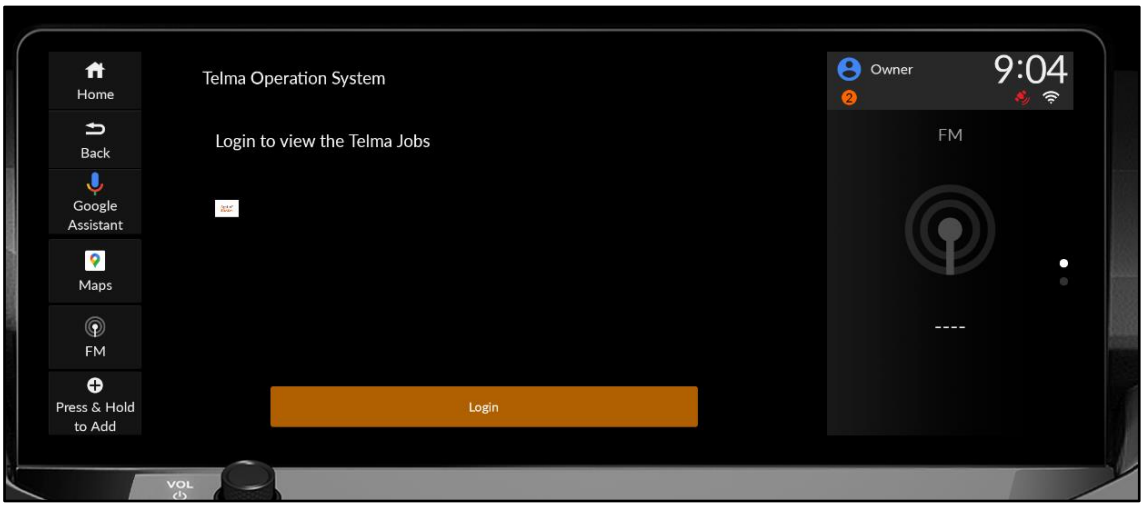

Figura 34. Aplicació en la pantalla d'un Honda

7.2.2 Volvo

Polestar és una submarca de Volvo, per tant, els models d'aquesta marca presenten una interfície i funcionalitat molt similars. Tot i això, hi ha algunes diferències subtils en la disposició de la pantalla i els elements gràfics que s'han pogut observar durant les proves.

Figura 35. Aplicació pantalla Login a Volvo

Figura 36. Pantalla Jobs Volvo

Figura 37. Pantalla Tasks Volvo

7.3 Avaluació del projecte

El projecte ha assolit els objectius proposats, ja que permet modificar l'estat de les proves i llegir les dades del vehicle segons el model i l'OEM seleccionat. Aquesta funcionalitat fa possible dur a terme proves d'assaig amb l'aplicació en qualsevol vehicle que utilitzi Android Automotive. El següent pas seria publicar l'aplicació a la *Play Store* d'Android Automotive, on Google haurà de revisar i aprovar l'aplicació segons els seus criteris i paràmetres.

7.3.1 Requisits funcionals

Per testejar els requisits funcionals, s'ha anat seguint una pauta de validacions i errors durant cada pantalla de l'aplicació.

La pauta a seguir són els guions dels requisits funcionals on:

- **Inici de sessió:** La pantalla d'inici de sessió mostra missatges de validació per informar si l'inici de sessió ha estat correcte o incorrecte.

- **Crides a l'API TELMA:** Les crides a l'API estan subjectes a validacions per garantir que es poden realitzar i enviar correctament. En cas d'error, l'operari té la possibilitat de tornar a la pantalla anterior per corregir la situació.

- **Lectura de dades del vehicle:** Per accedir a les dades del vehicle, l'usuari ha de concedir els permisos de seguretat necessaris.

Durant el desenvolupament, s'ha tingut en compte la gestió d'errors en cada etapa. En cas d'error, es mostra un missatge informatiu a l'usuari a través de la plantilla de missatges (Message Template), explicant clarament el problema i el seu context.

7.3.2 Requisits no funcionals

Els requisits no funcionals han estat considerats en tot moment durant el desenvolupament de l'aplicació. Aquests requisits no han presentat cap problema ni han estat una barrera al llarg del procés, garantint així un rendiment òptim i una experiència d'usuari fluida.

8 Avaluació dels Costos

8.1 Personal

En qualsevol projecte, és crucial identificar els rols necessaris per dur-lo a terme. En el cas del meu TFG, he assumit tots els rols necessaris, incloent-hi programador, dissenyador, tester i documentador.

El projecte es va dur a terme en un període de vint-i-dues setmanes, durant les quals es van definir objectius i abasts del projecte amb uns lliuraments esperats segons l'acord amb el meu tutor. Amb un conveni de TFG vigent des del maig fins a l'agost, la meva dedicació va ser de 32 hores setmanals amb un cost de 8 euros per hora.

Cost setmanal: *32 h/setmana * 8 €/hora = 256 €/setmana*

Cost personal del projecte: *256 €/setmana * 22 setmanes = 5.632 € generats en total*

8.2 Material

El projecte es basa principalment en el desenvolupament de programari, on tots els recursos Software necessaris són gratuïts.

Pel que fa al Hardware, s'utilitzaran els vehicles assignats a les proves, per la qual cosa que tampoc hi ha cost addicional en aquest àmbit, o bé, emuladors gratuïts integrats al software.

8.3 Taula de costos

El projecte, pensat per a tenir una continuïtat més enllà de la meva estada, ha estat iniciat per mi com a part del meu Treball de Final de Grau (TFG). Des del principi, es va plantejar amb un calendari de treball de quatre mesos i un pressupost assignat de 16.000 euros per cobrir les necessitats de personal. Aquest pressupost és el resultat de l'avaluació del cost que tindria contractar un equip de desenvolupadors per acomplir les mateixes tasques.

Per fer una comparativa clara, podem analitzar els costos associats al meu treball en contrast amb un escenari hipotètic en què un equip de desenvolupadors professionals hagués estat contractat per portar a terme el projecte. La Taula 2 mostra els costos proporcionats per Applus IDIADA per aquest projecte.

REQUESTED BUDGET SUMMARY	***2024***	***2025***	***TOTAL***
Hours costs	39000,00	12000,00	
Other Direct costs	750,00	2000,00	
Outsourced costs	0,00	0,00	
REQUESTED BUDGET per year (€)	39750,00	15000,0	54750,00
Additional information	**2024**	**2025**	**TOTAL**
Engineering hours per year (€)	1560,00	520,00	0,00

Taula 2. Taula de Costos proporcionada per Applus IDIADA pel projecte

Es poden fer càlculs comparatius entre el meu treball i el treball hipotètic sense la meva participació (2024):

El meu treball:

- He realitzat el projecte de manera individual, assumint tots els rols necessaris: programador, dissenyador, tester, i documentador.
- El projecte s'ha completat en un període de 22 setmanes, amb una dedicació de 32 hores setmanals a un cost de 8 euros per hora.
- Cost total: 256 €/setmana * 22 setmanes = 5.632 €.
- En aquest escenari, el projecte ha avançat de manera més lenta comparat amb un equip professional, però amb un cost significativament inferior.

Cas hipotètic amb un equip de desenvolupadors:

- Un equip de desenvolupadors professionals, amb més recursos i especialització, hauria pogut completar el projecte en menys temps i amb un volum de treball més gran.
- Cost total: 39.750,00 € només en personal per a l'any 2024.
- Aquest cost reflecteix una major inversió en temps i talent, amb l'avantatge de tenir més capacitat per gestionar la complexitat del projecte i complir amb terminis més ajustats.

8.4 Clients i zona de treball

A més, s'ha elaborat un pla de validació de mercats per identificar possibles interessats en l'ús d'aquesta aplicació. Ja es disposa de la validació d'interès per part de Hyundai Digital i dels operaris del departament d'IDIADA Durability / FOT. A més, la marca Renault – fleet Durability testing és un interessat hipotètic en aquesta tecnologia.

9 Legislació i Protecció de Dades

El desenvolupament d'una aplicació per a vehicles, com en el cas d'Android Automotive, està subjecte a diverses normatives i regulacions, especialment en relació amb la seguretat dels conductors i la protecció de les dades personals que s'han de tenir en compte alhora de la seva implementació. També s'ha tingut en compte els requisits de seguretat establerts per la mateixa empresa.

Legislació de Seguretat del Vehicle dins l'empresa

L'aplicació està dissenyada per funcionar exclusivament dins del recinte empresarial, ja que els vehicles utilitzats en les proves poden encara no haver sortit al mercat i, per tant, les dades associades són confidencials. A més, l'aplicació requereix connexió WIFI per realitzar les crides REST a TELMA, les quals només es poden efectuar si es troba connectat a la xarxa WIFI de l'empresa. Aquesta mesura garanteix que les dades siguin accessibles únicament pels empleats autoritzats.

Per a la realització de proves des de l'exterior, caldria connectar-se a la xarxa VPN de l'empresa, assegurant que totes les comunicacions i transferències de dades es realitzen de manera segura i protegida.

Seguretat del conductor

Per utilitzar l'aplicació, els operaris han d'atorgar permisos per a la lectura de dades del vehicle. Quan se sol·liciten aquests permisos, apareix un pop-up indicant quins són els permisos necessaris i l'usuari pot decidir si els atorga o no.

A més, l'aplicació ha estat desenvolupada tenint en compte les limitacions necessàries per garantir la seguretat del conductor, especialment per prevenir distraccions durant la conducció.

Control d'Accés

Establir un sistema de control d'accés és fonamental per assegurar qui pot accedir a quina informació dins de l'aplicació. Aquest control es realitza mitjançant Amazon Cognito, que permet definir rols i permisos de manera precisa, garantint que cada usuari tingui accés només a la informació i als recursos autoritzats segons la seva funció o necessitat.

L'aplicació també incorpora un sistema de Logout automàtic, on les dades de l'operari s'esborren de la memòria de l'aplicació un cop es desconnecta. Això evita l'emmagatzematge de tokens o dades personals, garantint que, en tornar a obrir l'aplicació, sigui necessari realitzar un nou inici de sessió, sense que es mantinguin dades personals o del vehicle anteriorment emmagatzemades.

L'aplicació ha estat dissenyada complint amb les regulacions de protecció de dades, assegurant que tota la informació personal dels usuaris sigui tractada de manera segura i en conformitat amb la normativa vigent, com el Reglament General de Protecció de Dades (GDPR).

10 Conclusions

En aquesta fase final del Treball de Fi de Grau, puc afirmar amb satisfacció que estic content amb els resultats obtinguts. El projecte ha estat una oportunitat per a posar en pràctica els coneixements adquirits durant els meus estudis i desenvolupar una aplicació funcional utilitzant tecnologies avançades. A través d'aquesta experiència, he millorat tant en la meva formació personal com professional, treballant en un entorn real de desenvolupament de Software com és Android Automotive.

Aquest projecte representa una fita significativa en la meva trajectòria, ja que ha estat la meva primera incursió en el món de l'enginyeria informàtica i del Software en un projecte d'aquesta envergadura, des de la conceptualització fins a la implantació. Aquesta experiència m'ha permès executar tasques fonamentals com la recopilació de requisits, el disseny gràfic de l'aplicació, la creació de diferents diagrames, i la implantació efectiva de l'aplicació.

Una de les parts més complexes del projecte ha estat configurar adequadament l'aplicació per garantir que funcioni de manera òptima dins l'entorn del vehicle, seguint estrictament les pautes definides per la seguretat del conductor. Aquesta ha estat una de les majors dificultats tècniques, però també una de les més valuoses a escala d'aprenentatge.

A més, aquest projecte m'ha permès familiaritzar-me amb noves tecnologies, com Android Automotive, que estan guanyant cada vegada més pes i popularitat en el món de l'automòbil. Aquesta plataforma està captant l'atenció de les marques més importants de vehicles, les quals busquen programadors capaços de desenvolupar aplicacions per implementar-les en els seus vehicles.

Un punt de millora per al futur seria continuar evolucionant l'aplicació per permetre que pugui efectuar més operacions sobre els Jobs, facilitant així encara més la feina dels operaris. Android Automotive és una eina relativament recent (creada el 2017), amb poca documentació disponible i un nombre limitat d'experts, fet que ha incrementat el meu interès en aquesta tecnologia i ha estat un dels factors clau a l'hora de triar aquest projecte.

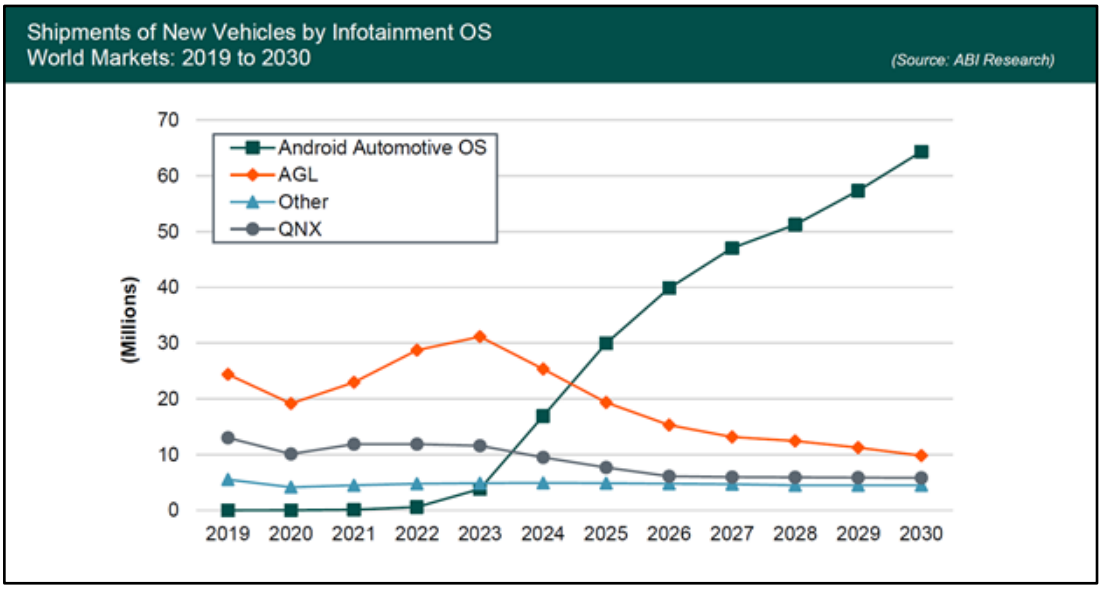

Figura 38. Vehicles amb el sistema operatiu d'Android Automotive en el futur

11 Recursos Utilitzats

11.1 Software

Per al desenvolupament de l'aplicació Android Automotive, s'ha utilitzat el framework **Android Studio** en la seva versió *Jellyfish | 2023.3.1*.

Aquesta eina ha estat fonamental per a la codificació, depuració, i testatge de l'aplicació en un entorn professional. A més, per poder executar l'aplicació en un emulador, ha estat necessari disposar d'un compte de Google per accedir als serveis de Google, com la Play Store, i per descarregar un navegador web com **Vivaldi**, requerit pel procés de Login.

Figura 39. Logotip Android Studio

Per a la programació de codi, s´ha utilitzat **Java 17.0.11 2024-04-16 LTS.**

Figura 40. Logotip Java

Les reunions setmanals de seguiment del projecte, tan empresarials com amb el tutor universitari, s'han realitzat mitjançant MS Project, una eina clau per a la planificació i gestió del projecte.

A més, s'han utilitzat altres programes gratuïts per a la creació de diagrames, com Draw.io i Eraser, que han estat essencials per documentar visualment els processos i estructures del projecte.

Per a la gestió del projecte i control de versions, s'ha utilitzat GitLab, que ha proporcionat un repositori segur i eficaç per a l'emmagatzematge i col·laboració en el codi font.

11.2 Hardware

Durant el desenvolupament de l'aplicació, es va utilitzar un ordinador amb **Sistema Operatiu Windows 11**. Aquest ordinador va ser equipat amb un emulador **HMI** (Human-Machine Interface) del vehicle **Polestar 2** dintre d'Android Studio per simular les interaccions i comportaments de l'aplicació en un entorn controlat.

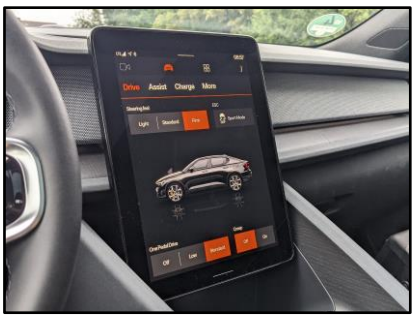

Figura 41. HMI del vehicle Polestar 2

Per al testatge en un entorn més proper a la realitat, es va utilitzar un vehicle **Polestar 4** amb tecnologia **Android Automotive**. Aquest vehicle va servir per validar la funcionalitat d'Android Automotive. No obstant això, cal destacar que l'aplicació encara no s'ha pogut testejar en un vehicle real operant en el recinte empresarial. Això és perquè l'aplicació ha de ser validada prèviament per Google, i a més, l'empresa està en procés de discutir els detalls amb un OEM (Original Equipment Manufacturer) per garantir la seva compatibilitat i conformitat amb els estàndards de la indústria.

Aquesta validació és un pas crucial, ja que assegura que l'aplicació compleix amb totes les normatives i requisits de seguretat, tant per part de Google com dels fabricants de vehicles, abans de ser implementada en entorns reals. Fins que no es completin aquests processos, l'aplicació es mantindrà en un estat de prova intern, limitat a l'ús dins l'empresa.

Figura 42. Model Polestar 4

11.3 Bibliografia

[1] Pàgina Web https://developer.android.com/training/cars?hl=es-419. [Que és AA] abril 2024

[2] Pàgina Web ¿Qué es Android Automotriz? | Android Open Source Project [Definició AA] abril 2024

[3] Pàgina Web https://lembergsolutions.com/blog/android-automotive-os-app-development-insights [Limitacions de AA] abril 2024

[4] Pàgina Web https://developer.android.com/training/cars/apps/automotive-os?hl=es-419#automotive-module [Projecte AA] abril 2024

[5] PDF eur-lex.europa.eu/legal-content/ES/TXT/PDF/?uri=CELEX:32016R0679 [Legislació] maig 2024

[6] Pàgina Web Agencia Española de Protección de Datos | AEPD [Legislació] maig 2024

[7] Vídeo Android Automotive Architecture Explained in Detail 2023 (youtube.com) [Arquitectura AA] maig 2024

[8] PDF XVIII CONVENIO COLECTIVO ESTATAL DE EMPRESAS DE CONSULTORÍA, TECNOLOGÍAS DE LA INFORMACIÓN, Y ESTUDIOS DE MERCADO Y DE LA OPINIÓN PÚBLICA [Costos] maig 2024

[9] Pàgina Web Bring your Android App into your car — Part 1 : What you should know | by Loïc Teyssier | Medium [Android Automotive project] maig 2024

[10] Pàgina Web Compila una app de Internet de las cosas | Android Developers [AA IoT] maig 2024

[11] Pàgina Web How to setup an Automotive Android Emulator | by Malte Wolfcastle | Medium [Polestar 2 emulador Android Studio] maig 2024

[12] Pàgina Web Design for Driving | Google for Developers [Design for Driving] maig 2024

[13] Pàgina Web java - How do I call REST API with bearer token from an Android app? - Stack Overflow [Tokens AWS Cognito] juny 2024

[14] Pàgina Web Login endpoint - Amazon Cognito [API Cognito] juny 2024

[15] Pàgina Web Logout endpoint - Amazon Cognito [API Cognito] juny 2024

[16] Pàgina Web Retrofit (square.github.io) [Retrofit2] juny 2024

[17] Pàgina Web Eraser [Diagrames] juliol 2024

[18] Pàgina Web Android Automotive #4 — Consuming the VHAL in a managed application | by ImaginationOverflow | Medium [VHAL Metadata] juliol 2024

[19] Pàgina Web java - Retrofit 2.0: how to generate pojo class for dynamic objects - Stack Overflow [POJO] juliol 2024

[20] Pàgina Web Grid template | Design for Driving | Google for Developers [Grid Template UI] juny 2024

[21] Pàgina Web List template | Design for Driving | Google for Developers [List Template] juny 2024

[22] Pàgina Web Polestar Developer Portal – automotive emulator | Polestar [Polestar developers] julio 2024

[23] Pàgina Web First steps with Android Automotive Car API | by Filip Guzy | siili_auto | Medium [API Car] agost 2024

[24] Pàgina Web Software Icons & Symbols (flaticon.com) [Software icons] juliol 2024

[25] Pàgina Web Car | Android Developers [Car] agost 2024

[26] Pàgina Web VehiclePropertyIds | Android Developers [VehiclePropertyIds] agost 2024

[27] Pàgina Web service/AndroidManifest.xml - platform/packages/services/Car - Git at Google (googlesource.com) [Seguretat en el Manifest] agost 2024

[28] Pàgina Web Cómo probar apps de Android para vehículos | Android Developers [Test apps en vehicles] agost 2024

[29] Pàgina Web ¿Qué es Android Automotriz? | Android Open Source Project [Apps per vehicles] agost 2024

12 Annexos

12.1 Posada en marxa

Sistema Operatiu

Per poder executar l'aplicació tal com es va dissenyar i complir amb els objectius del projecte, és necessari disposar d'un vehicle equipat amb la tecnologia **Android Automotive**. Aquesta és la manera òptima de garantir que l'aplicació funcioni correctament en un entorn real.

En cas de no disposar d'un vehicle real, es pot executar l'aplicació utilitzant un dispositiu amb Windows 11. Per això, és necessari tenir instal·lats Java i Android Studio. A més, cal configurar un emulador HMI (Human-Machine Interface) del vehicle per simular les condicions de funcionament del sistema.

Configuració

Per configurar l'aplicació de manera que es pugui executar correctament amb Android Automotive, és necessari crear un mòdul específic per a Automotive dins d'Android Studio. Això implica verificar i descarregar la imatge del sistema Automotive al **SDK Manager** a la secció de **Platforms**.

Durant la creació del projecte, s'ha utilitzat l'**API 29: Android 10.0 (Q)**, que s'ha configurat com la base per al desenvolupament de l'aplicació. Aquesta configuració assegura la compatibilitat amb la majoria dels vehicles que utilitzen Android Automotive.

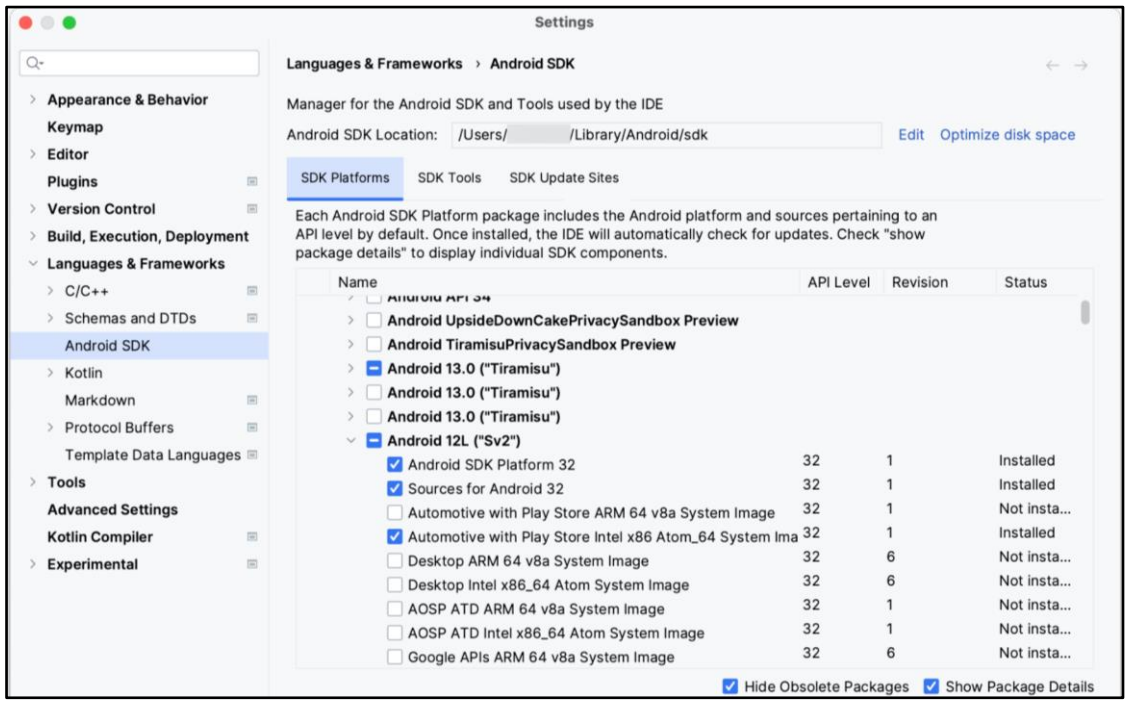

Figura 43. Configuració del SDK Platforms perquè funciona amb Automotive

12.2 Setup del projecte

En el següent README.md s'explica detalladament com configurar el codi d'un projecte d'Android Automotive:

IDIADA/README.md at master · mrallfredii/IDIADA (github.com)

12.3 Execució

Si estem en el vehicle amb Android Automotive, es pot accedir a l'aplicació directament des de la interfície HMI del vehicle, on l'aplicació haurà estat prèviament instal·lada.

Si es fa servir un dispositiu Windows, cal accedir al projecte a Android Studio: **IDIADA_AA**.

Primer, obre el **Device Manager** dins d'Android Studio i selecciona el dispositiu configurat, que actua com a emulador del sistema HMI del vehicle.

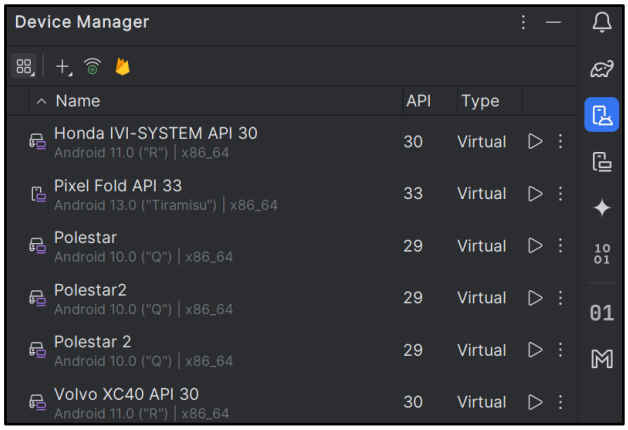

Figura 44. Device Manager

Després, executa el projecte seleccionant l'opció **Run**.

Figura 45. Run projecte

Per obtenir una guia detallada sobre la configuració i execució de l'emulador Polestar2 a Android Studio a Device Manager, pots consultar el següent enllaç:

How to setup an Automotive Android Emulator | by Malte Wolfcastle | Medium