App Lector OBD

https://github.com/Hug0Lz/Gestor-Coches-App.git

Nome Alumno/a: **Hugo López Carballo**

Curso: ***2º DAM***

Módulo: ***Proxecto Final Ciclo***

Contido

Tabla de contenido

Contido 2
1. Introducción 3
2. Obxectivos 4
3. Situación previa 4
4. Tecnoloxías empregadas 4
5. Solución proposta 4
6. Planificación do proxecto 5
Planificación do proxecto cun diagrama de Gantt 5
Descripción do modelo de desenvolvemento de software a implementar. 6
Análise do proxecto 7
Deseño do proxecto: 8
Presuposto completo (Hardware software e recursos humans) 15
7. Desenvolvemento e execución 16
Diagrama de Despregamento 16
Descripción xeral do funcionamento da aplicación. 16
8. Conclusións e reflexións 17
9. Bibliografía e Webgrafía 18
10. Anexo I – Manual técnico de instalación ou posta en marcha 19

1. Introducción

Co avance das tecnoloxías, os vehículos fanse máis complexos cada xeración. Máis sensores que permiten facer mellores diagnósticos, máis sistemas de seguridade... Isto fixo necesario avanzar na creación de sistemas electrónicos para os vehículos que os melloraron en moitos aspectos. Dende a mecánica con avances coma a inxección electrónica, a incorporación de tecnoloxías aplicadas a seguridade coma os controles de tracción electrónicos(ESP). Isto fixo necesario a creación de unidades que procesaran toda a información e xestionaran os diferentes sensores e actuadores, tendo lugar a aparición de centralitas (ECU). Procesadores centralizados que controlan diferentes aspectos dos vehículos. A día de hoxe, nun mundo tan dixitalizado, fixéronse comúns as melloras a nivel de interface de usuario e infoentretemento no vehículo, con cada vez pantallas máis grandes e interfaces máis elaboradas. Tamen foi así no caso das centralitas. En coches máis antigos podemos atopar centralitas simples para o motor, a transmisión e sistemas de seguridade. Pero a día de hoxe podemos contar con centos de centralitas dedicadas a diferentes aspectos do vehículo: sistemas de seguridade activa, controles adaptativos de cruceiro, sistemas de infoentretemento...

Co aumento da complexidade destos sistemas nace a necesidade de novas interfaces de comunicación cos vehículos, tanto para o seu mantemento como para a súa xestión. Isto lévanos a ver a necesidade dunha aplicación que nos permita interactuar co noso vehículo, vendo en directo diferentes parámetros e datos do mesmo, ademáis de ter conta dos seus mantementos e levar un rexistro da manutención.

A partir do ano 1991 incorporouse OBD-I (On Board Diagnostics) en tódolos vehículos, sentando as bases dos primeiros sistemas de monitorización para algúns compoñentes dos vehículos coma os controladores de emisións. Non obstante, OBD-I non era un estándar universal, e os fabricantes facían implementacións propias e con diferentes nivel de exactitude e veracidade a hora de mostrar datos.

Máis adiante, coa chegada de novas normas de emisións e o aumento dos sistemas informáticos nos vehículos, nace a segunda xeración do sistema: OBD-II. Este novo estándar, introducido no 1996 en Estados Unidos permite detectar fallos eléctricos, químicos e mecánicos que poidan afectar as emisións do vehículo.

Os vehículos gardan nas ECU un rexistro destes fallos, permitindo a un mecánico revisar os valores dos sensores e facendo máis doado o diagnóstico de un erro no funcionamento normal dos mesmos.

Tamén permiten, coas interfaces adecuadas, comunicarse co vehículo de modo que se poidan editar valores. Isto permite, dende a codificación de novas chaves, ata o desbloqueo de características ocultas no vehículo ou reservadas a gamas superiores. Tamén se poden facer edicións no sistema motor, editando a mezcla de aire/combustible e personalizando o funcionamento dos inxectores. Claro que, por motivos de seguridade e intereses propios das marcas, necesítanse protocolos personalizados para a comunicación con certas centralitas, especialmente para a súa edición

Pero, a partir do ano 2000 para vehículos gasolina e 2003 para vehículos diésel, oblígase a introducción de portos OBD-II nos vehículos comercializados en Europa baixo o estándar EOBD (European On-Board Diagnostics) que será a versión europea do estándar OBD-II. Este estándar permitirá, con un sistema de diagnosis, comunicarse con calquera vehículo para leer erros e realizar diversas probas dos sistemas do vehículo.

2. Obxectivos

Aínda que se pode profundizar moito nos diferentes protocolos OBD e os estándares compatibles, o obxectivo da aplicación é dar acceso ao usuario a datos básicos de funcionamento do vehículo tales como a medición de temperatura ambiente, temperaturas de entrada do aire, temperatura do refrixerante e do aceite, porcentaxe de aceleración e frenada, velocidade do vehículo e revolucións por minuto, entre outras.

Coma os protocolos empregados en cada vehículo varían, a aplicación centraráse non tanto en poder mostrar moitos datos, senón nos máis cruciais.

Ademáis, outro dos pilares da aplicación será proveer dunha interface na que apuntar os mantementos realizados no vehículo, así como crear recordatorios para futuros mantementos. Por exemplo, recordar o cambio do aceite unha vez o ano, ou a revisión dos frenos cada 6 meses.

3. Situación previa

No mercado xa existen máquinas de diagnose que leen portos OBD, pero a maioría son equipos profesionais de alto custo. Para un usuario normal, que só queira saber algo máis do estado do seu vehículo, podería bastar cunha interface máis sinxela. Por iso, óptase por empregar un sistema baseado no chip ELM327, un procesador de baixo custo e open source que nos permitirá comunicarnos co vehículo mantendo o presuposto axustado.

Tamén temos aplicacións de notas ou recordatorios que nos poderían permitir tomar conta dos mantementos, pero coa nosa aplicación buscamos centralizar todo nunha sola app.

4. Tecnoloxías empregadas

O desenvolvemento da aplicación, xa que se centra nos dispositivos móbiles iOS, estará baseada no framework de SwiftUI desenvolto por Apple, presentado no seu WWDC de 2019. Trátase dunha alternativa máis sinxela a anterior ferramenta UIKit, introducindo melloras coma a sintaxe declarativa. Tamén dan facilidades para a adaptación das aplicacións a diferentes interfaces e dispositivos: iOS, macOS, watchOS e tvOS.

Para a conexión Bluetooth, empregaráse a librería *SwiftOBD2*, que emprega o framework de apple CoreBluetooth que permite a conexión con dispositivos BLE (Bluetooth Low Energy).

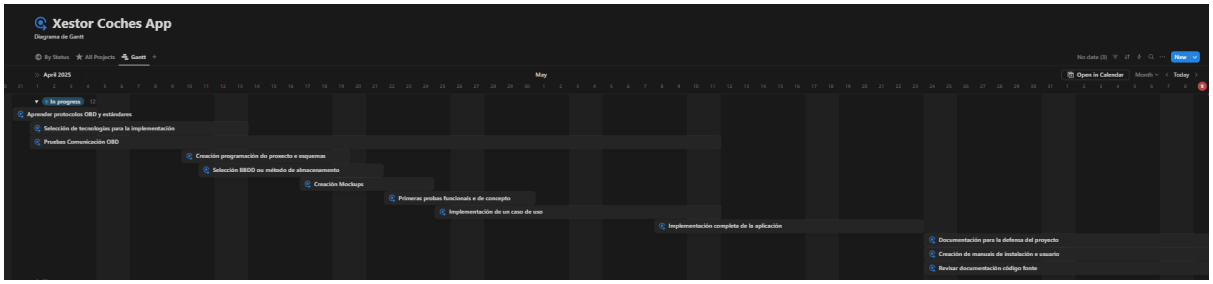
Para o almacenamento de datos empregaranse arquivos Json.

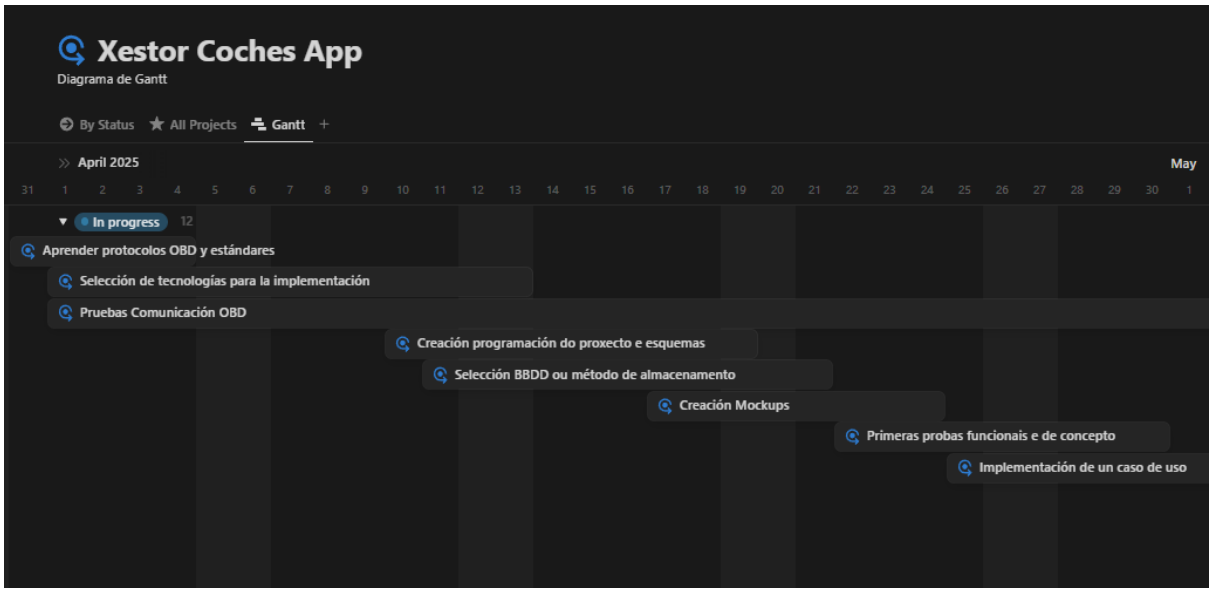
5. Solución proposta

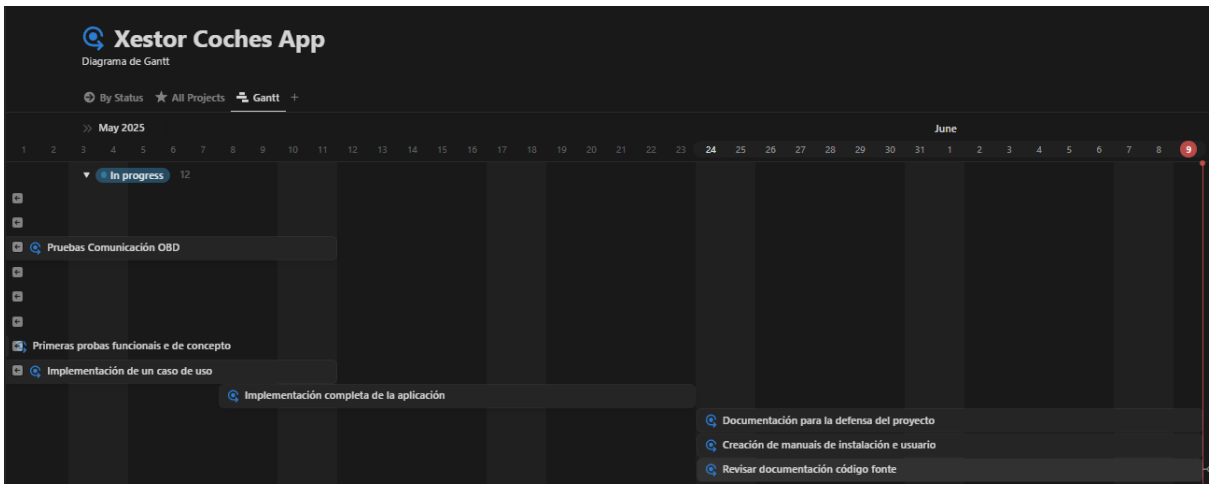
A proposta, en poucas palabas, trátase dunha aplicación sinxela na que revisar diferentes parámetros do coche e, ademáis, levar conta dos mantementos do mesmo. Ao basearse nun sistema open source asequible, trátase dunha alternativa de baixo custo para o usuario e que pode ser moi útil, centralizando todo o relacionado co vehículo nunha mesma aplicación

6. Planificación do proxecto

Planificación do proxecto cun diagrama de Gantt







Descripción do modelo de desenvolvemento de software a implementar.

O proxecto seguiu un modelo de desenvolvemento incremental. Este enfoque permitiu introducir funcionalidades de forma progresiva, engadindo novas vistas, funcionalidades e compoñentes en pequenas fases e mellorando os xa existentes. En cada iteración realizáronse melloras e revisións para solucionar erros detectados e para adaptar o deseño as necesidades finais da aplicación.

Este modelo resulta especialmente útil en proxectos onde non está clara a estrutura final do mesmo desde o inicio, permitindo flexibilidade e unha mellor xestión dos cambios.

Tamén permitiu o desenvolvemento e mellora paralela de diferentes partes da aplicación, como a xestión de datos de mantementos, a interface de usuario e a integración co OBD.

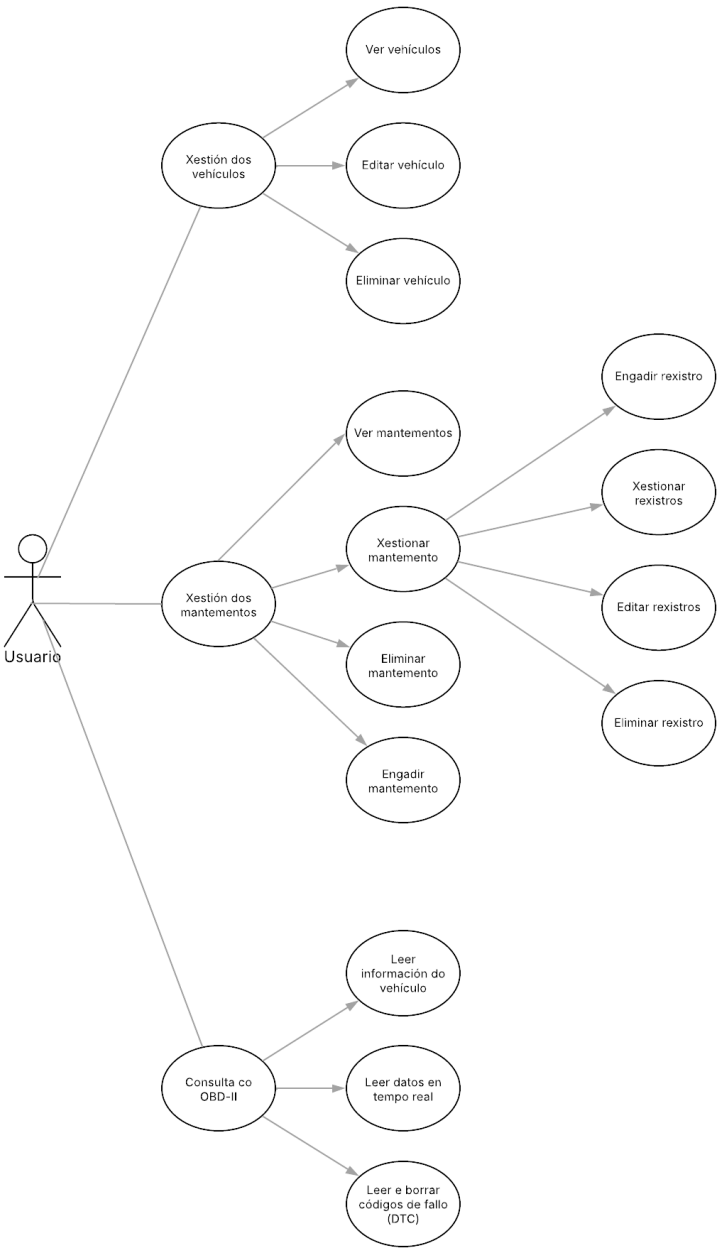
Cada incremento centrouse nun obxectivo específico, como a conexión co OBD, lectura de datos do OBD, persistencia de datos de mantementos en ficheiros JSON ou creación de interfaces para interactuar con estos rexistros.

O modelo incremental foi especialmente interesante para un proxecto destas características. Nun proxecto xestionado por unha sola persoa facíase excesivo empregar outras alternativas como a metodoloxía áxil SCRUM, que sería máis tediosa de implementar.

Este proceso iterativo permitiu un desenvolvemento igualmente áxil, favorecendo a experimentación de diferentes métodos de conexión e a mellora continua da aplicación e das súas características.

Análise do proxecto

Diagrama de casos de usos.

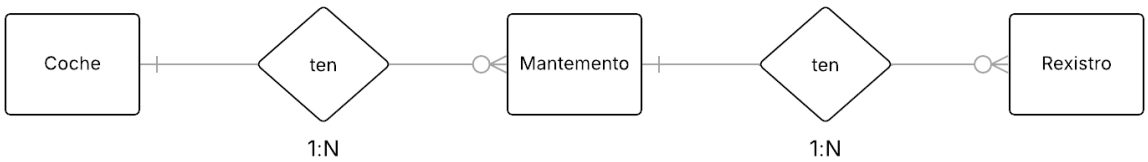


Universo de discurso da base de datos.

O universo de discurso da base de datos da aplicación está composto polos seguintes elementos principais:

- Garaxe: almacena un conxunto de vehículos (coches) rexistrados polo usuario na aplicación
- Coche: contén información básica do vehículo (marca, modelo, combustible...) e garda os seus mantementos
- Mantemento: representa un tipo de operación realizada no vehículo (un cambio de aceite, revisión dos freos...)
- Detalle do mantemento: representa unha execución concreta do mantemento ao que se relaciona, podendo gardar información adicional da aplicación coma a kilometraxe a que se fixo ou calquera apunte que o usuario vexa necesario.

Diagrama de Entidad-Relación.



Deseño do proxecto:

Modelo relacional da base de datos.

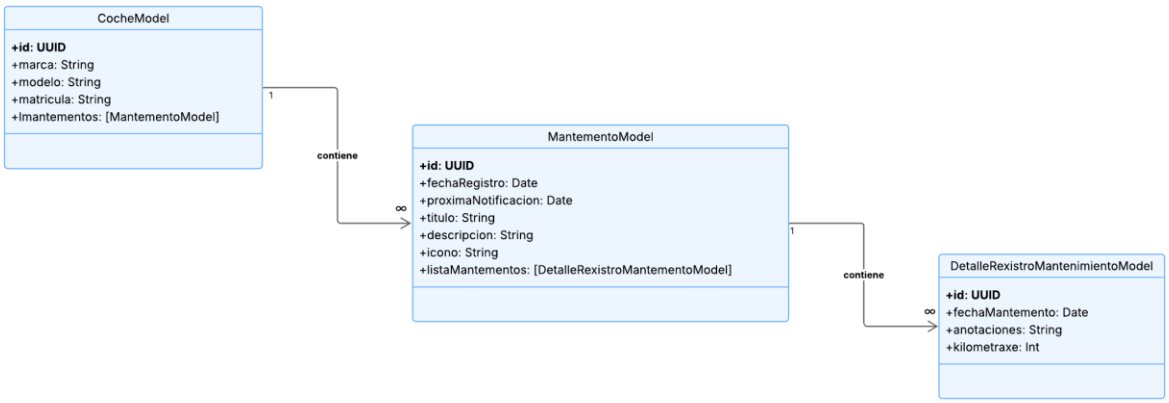
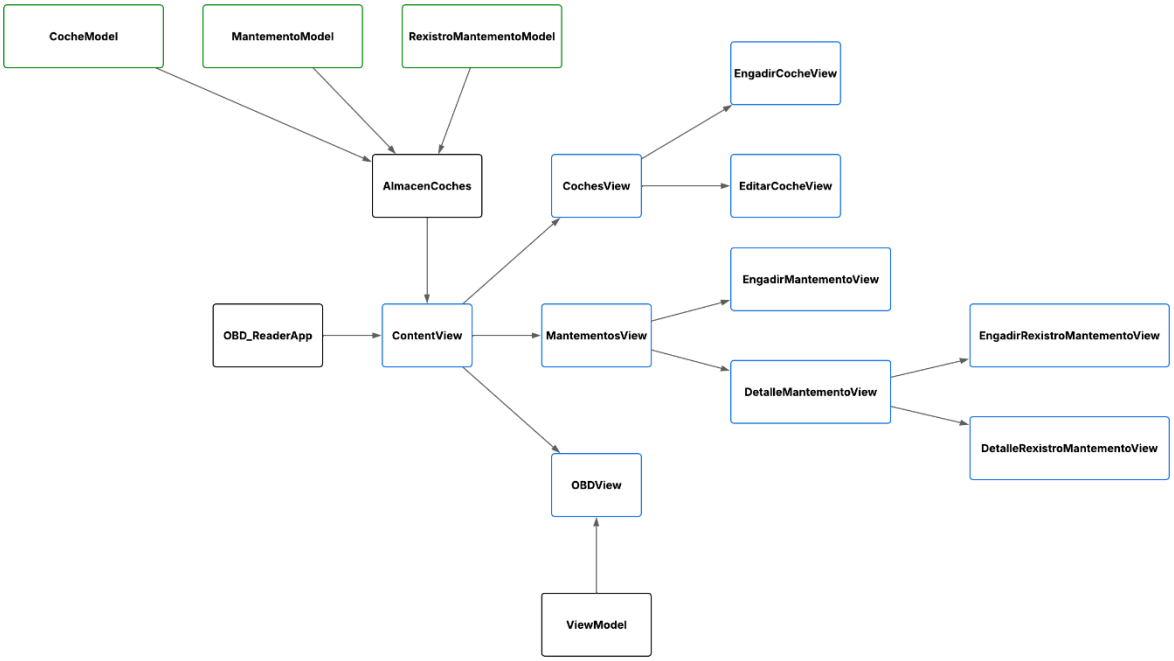


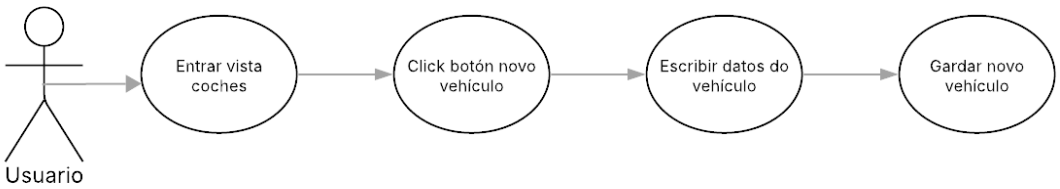
Diagrama de clases.

Diagrama das clases e despois diagrama das vistas

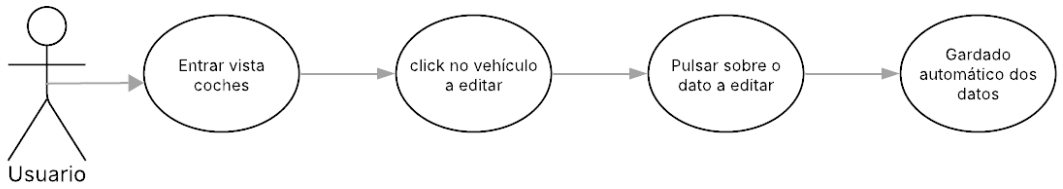


Diagramas de fluxo de cada caso de uso.

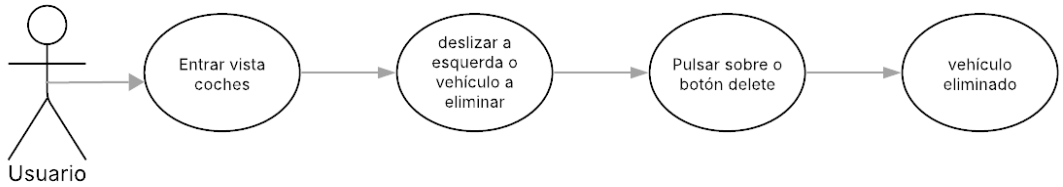
Engadir un vehículo



Editar un vehículo

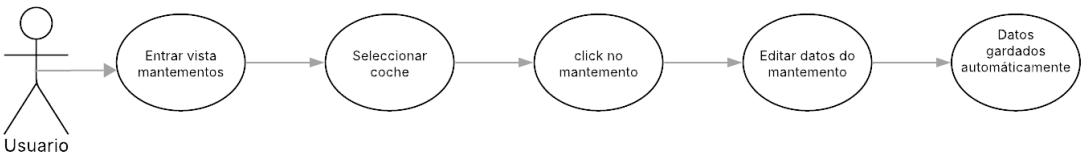


Eliminar un vehículo

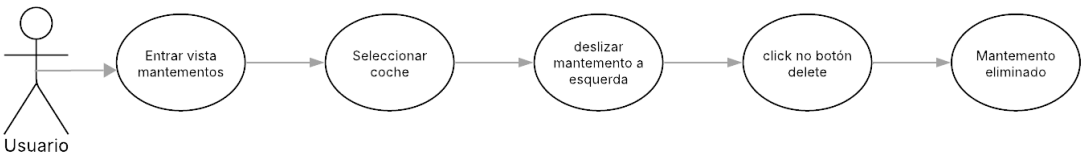


Engadir un mantemento

Editar un mantemento



Eliminar un mantemento

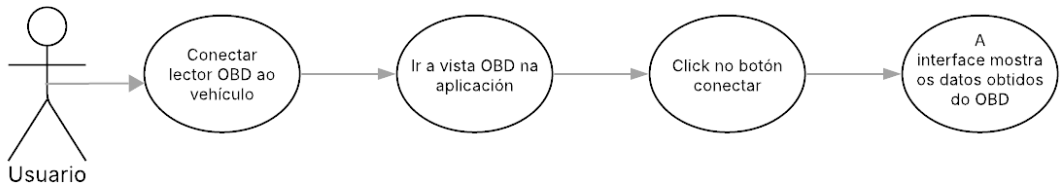


Engadir un rexistro

Editar un rexistro

Eliminar un rexistro

Conectarse ao OBD

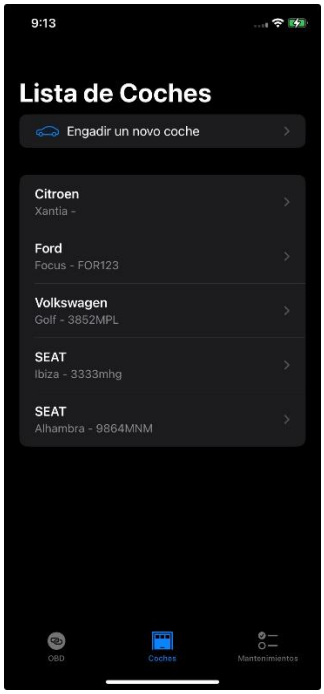


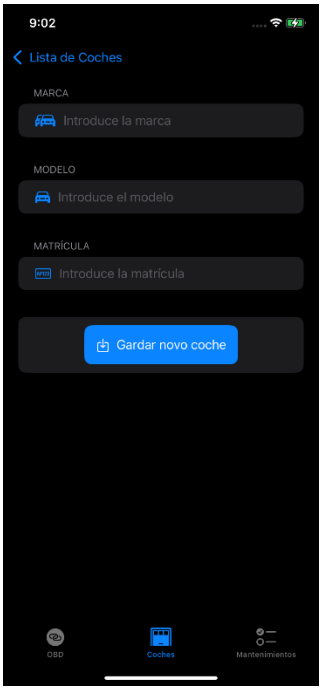
Mockups da interface.

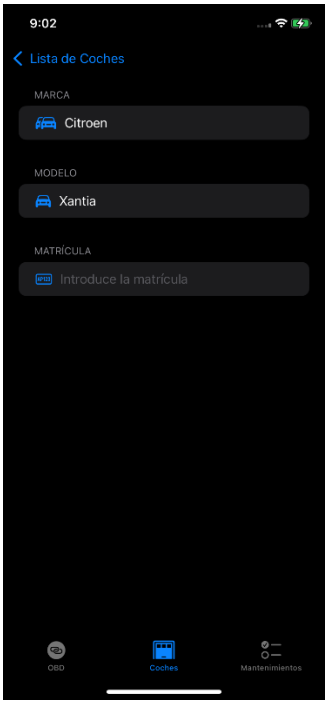
Interfaz OBD

- Aquí temos a vista donde se mostrarán os datos do OBD. Tamén dispomos dun modo demo para simular os datos que se verían nos diferentes sensores

Xestión dos vehículos

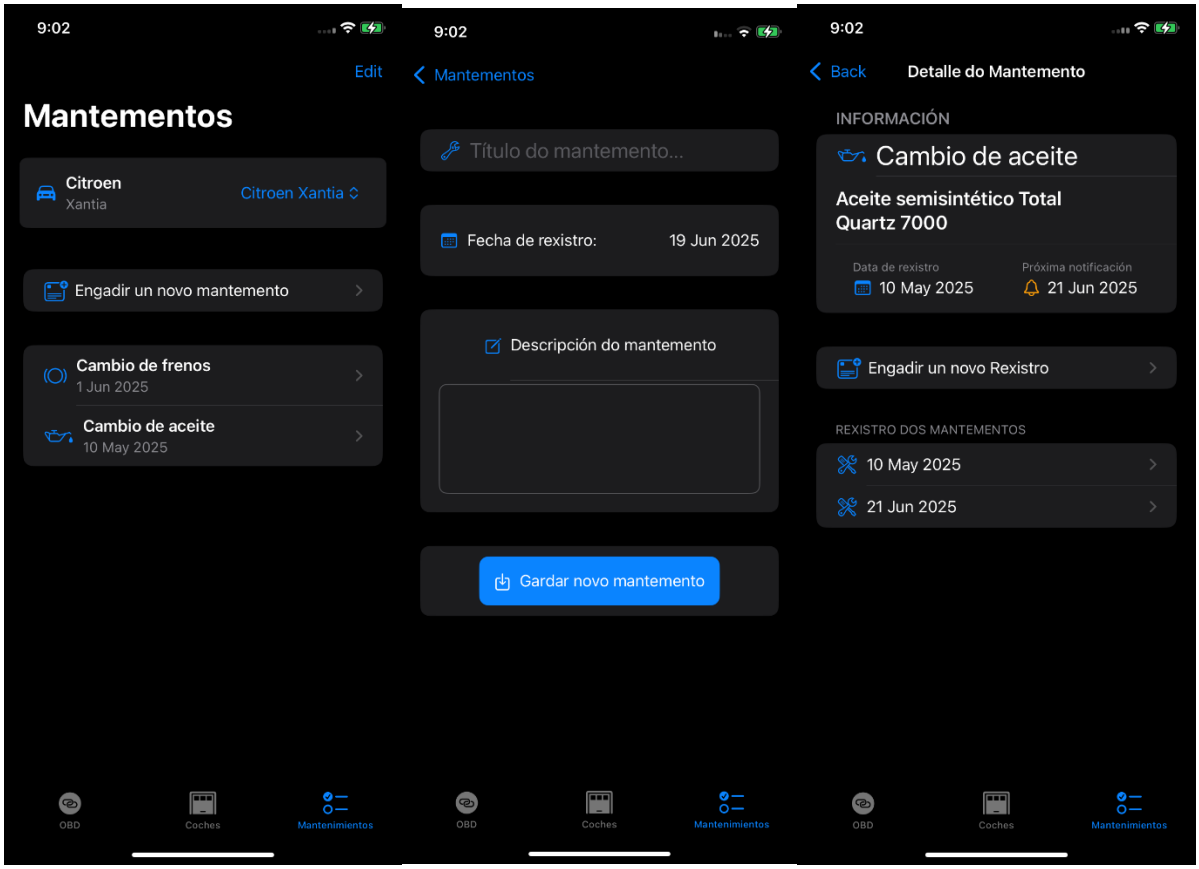






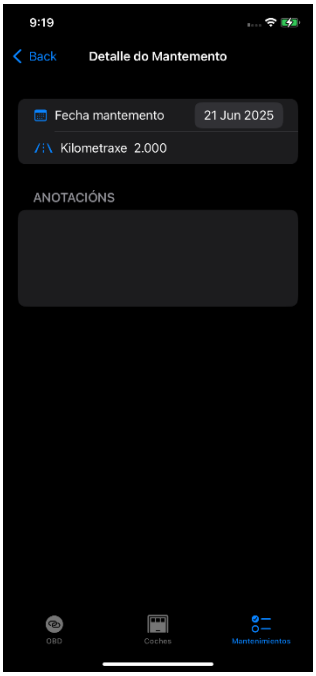
Na primeira vista vemos o listado dos vehículos rexistrados. Na segunda a vista para engadir un vehículo, e na terceira a vista para editar un vehículo xa rexistrado

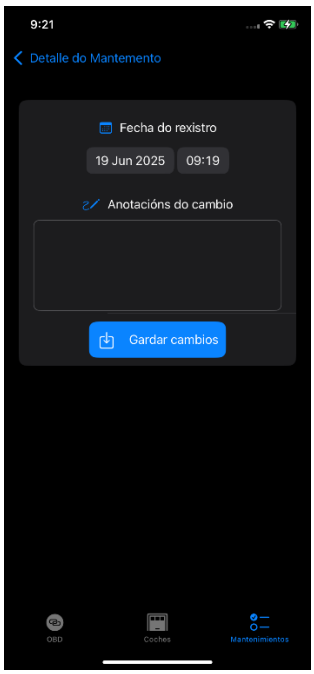
Xestión dos mantementos



- Aquí vemos as vistas de mantementos. Na primeira, o listado dos mantementos. Na segunda, a vista para engadir un novo mantemento. Na terceira, o detalle que nos aparece ao facer click nun mantemento.
- Tamén podemos editar os rexistros de cada mantemento:

Rexistros Mantemento





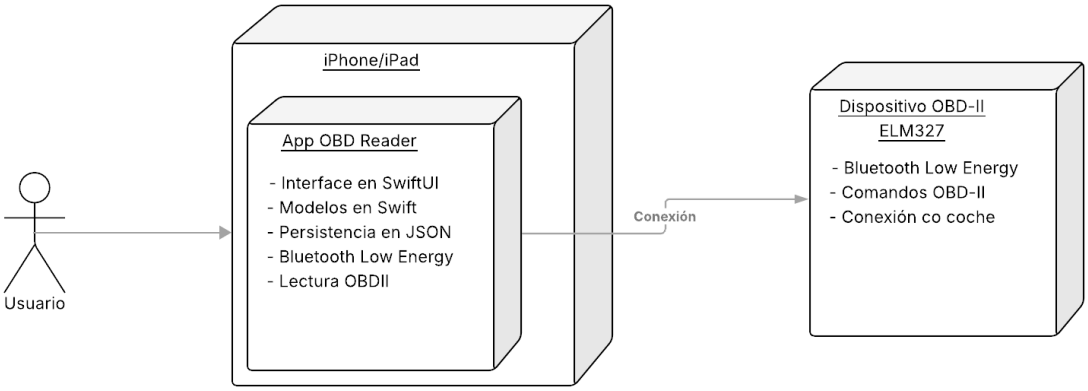
- Aquí temos as vistas para os rexistros do mantemento, que nos permiten

Presuposto completo (Hardware software e recursos humans)

Nome	Tipo	Prezo
Macbook Pro i7	Hardware	594,99€
iPhone XR	Hardware	174,00€
Lector OBD ELM327	Hardware	8,99€
Framework SwiftUI e XCode	Software	0€
Developer SwiftUI	Recursos humáns	500€

7. Desenvolvemento e execución

Diagrama de Despregamento



Descripción xeral do funcionamento da aplicación.

- A aplicación desenvólvese con SwiftUI para traballar nun entorno iOS. O seu funcionamento baséase na lectura de datos en tempo real dun dispositivo OBD-II conectado o vehículo, ademais da xestión de mantementos dos vehículos do usuario.
- A estrutura da aplicación emprega un patrón MVVM (Modelo – Vista – ModeloVista) que serve para separar a lóxica do programa da interfaz de usuario, empregando 3 elementos clave:
    - Modelos: representan a capa de datos e lóxica de negocio. Conteñen a información
    - Vista: Representa a información a través de elementos visuais e permite interactuar coa mesma.
    - Modelo de vista: é un intermediario entre o modelo e a vista, que contén a lóxica para a presentación dos datos.
- A persistencia de datos realízase mediante un ficheiro Json local gardado na aplicación, sen requerimento de bases de datos adicionais ou conexión a outros servidores
- Durante o desenvolvemento resolvéronse varios retos técnicos:
    - Lectura de datos OBD-II por BLE: empregouse un lector ELM327 compatible co protocolo Bluetooth Low Energy ao que nos conectamos dende a app para enviar comandos estándar do protocolo OBD-II
    - Decodificación e procesamento dos datos: deseñouse o proceso de decodificación dos datos obtidos do dispositivo, decodificando os seus valores para mostrar parámetros coma as RPMs ou a temperatura do motor

- Persistencia de datos: implementouse unha solución lixeira baseada na codificación e decodificación de modelos Swift empregando o protocolo *Codable*. Isto permitiu aforrarse o uso de outras ferramentas propias da linguaxe máis complicadas e tediosas como CoreData, deixando un modelo máis simple.
- Xestion dos mantementos: deseñouse un sistema onde gardar cada mantemento cos seus rexistros, de maneira que se poidan asociar datas e kilometraxe a cada iteración do mantemento.
- Deseño adaptativo con SwiftUI: toda a interface da aplicación desenvolveuse con vistas reactivas e listas dinámicas vinculadas a modelos observables, o que permite que en todo momento se manteña actualizada a información mostrada cos datos gardados, ademais de adaptarse a diferentes tamaños de pantalla (iPhone e iPad)

8. Conclusións e reflexións

A realización deste proxecto supuxo un reto técnico e persoal. Xa antes de decidir plataformas e tecnoloxías, a dúbida estaba nun lugar: a conexión co hardware. Isto implicou non só aplicar os coñecementos adquiridos ao longo do ciclo, senón tamén buscar e analizar moita información sobre o funcionamento dos protocolos e a maneira de empregalos.

Decidiuse, polo tipo de aplicación e os seus obxectivos, orientarse a unha plataforma móbil. Tras valorar varias alternativas (coma Android Studio ou Flutter) optouse pola vista en clase: o ecosistema Apple. Aínda que xa tíñamos adquirido certos conceptos sobre o desenvolvemento de interfaces, este proxecto requeriu de novas ferramentas e librarías. Por exemplo, o uso de *CoreBluetooth* para a conexión co dispositivo BLE ou SwiftOBD2 e SymbolPicker para facilitar a conexión OBD e a selección de iconas por parte do usuario.

Tamén foi necesario aprender sobre o funcionamento do protocolo OBD dos vehículos, os seus conceptos clave e os distintos tipos de comandos soportados dependendo dos estándares empregados..

Foi preciso investigar o funcionamento do chip ELM327 para entender que comandos podía enviar o dispositivo, e como había que conectarse tanto o dispositivo coma despois enviar a orde para que o lector se conectara o vehículo.

En definitiva, este proxecto permitiume profundizar moito na conexión de periféricos hardware ao noso software. Tamén me ensinou a xestionar o desenvolvemento de software de principio a fin, dende a análise inicial e toma de decisións tecnolóxicas ata o deseño da interface, o tratamento dos datos e a documentación do traballo realizado. Inda que se poderían valorar moitas melloras (refinado das vistas e adición de novas funcionalidades) e incluso outras aproximacións aos problemas que se pretendían solucionar (coma o uso de outras plataformas, ou a creación dunha aplicación para ordenador), quedo satisfeito coa aplicación e considéroa un primer paso. Unha gran base para seguir desenvolvendo e mellorando, ou se cadra a inspiración de novos proxectos.

9. Bibliografía e Webgrafía

Formato APA

Wikipedia. (s.f.). *OBD*. Wikipedia, La enciclopedia libre. https://es.wikipedia.org/wiki/OBD

Elm Electronics. (2009). *ELM327DS: ELM327 – OBD to RS232 Interpreter* (versión 1.4b) [PDF]. SparkFun. https://cdn.sparkfun.com/assets/learn_tutorials/8/3/ELM327DS.pdf

Elm Electronics. (2005). *ELM327 AT Commands Summary Sheet* [PDF]. SparkFun. https://cdn.sparkfun.com/assets/4/e/5/0/2/ELM327_AT_Commands.pdf

Apple Inc. (s.f.). *Apple Developer*. https://developer.apple.com/

Konteh, K. (2024). *SwiftOBD2* [Repositorio en GitHub]. https://github.com/kkonteh97/SwiftOBD2

Konteh, K. (2023, abril 10). *Build an OBD2 car diagnostics app with Swift*. Medium. https://medium.com/@kkonteh97/build-an-obd2-car-diagnostics-app-with-swift-02e91b6f828e

Xnth97. (2024). *SymbolPicker* (Versión más reciente) [Repositorio en GitHub]. https://github.com/xnth97/SymbolPicker

10. Anexo I – Manual técnico de instalación ou posta en marcha

- Para a instalación, deberáse clonar o repositorio
- Tras clonar o repositorio, abrímolo con Xcode
- Abrimos o proxecto con Xcode e xa podemos velo dende o simulador
- Para probar a funcionalidade completa, é interesante dispor dun dispositivo iPhone, a fin de probar as capacidades do lector OBD. Para configurar o simulador, podemos seguir o tutorial da páxina de Apple