Joel Haapaniemi

OBD2 Diagnostiikka Swift -sovellus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja Viestintätekniikan tutkinto

Insinöörityö

4.9.2023

Tiivistelmä

Tekijä:	Joel Haapaniemi
Otsikko:	OBD2 Diagnostiikka Swift -sovellus
Sivumäärä:	31 sivua
Aika:	4.9.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja Viestintätekniikan tutkinto
Ammatillinen pääaine:	Ohjelmistotekniikka
Ohjaajat:	Opinto-ohjaaja Keijo Länsikunnas

Avainsanat: elm327, obd2

Tämän opinnäytetyön tausta ja tavoite oli luoda sovellus puhelimille, joissa on iOS-käyttöjärjestelmä, jolla voitaisiin luoda langaton bluetooth-yhteys henkilö-auton obd2-järjestelmään. Yhteyden luomiseksi hyödynnettiin bluetooth low energy obd2 elm327 -adapteria. Tämän adapterin tulee olla kytkettynä henkilö-auton obd2-porttiin, jotta applikaatio saa dataa auton obd2-järjestelmästä hyödynnettäväksi sovelluksessa. Työllä ei ollut varsinaista asiakasta tai tilaajaa, vaan ideana oli luoda sovellus julkaistavaksi Applen AppStoreen.

Työn toteutus alkoi tutustumalla ohjelmointiympäristöön ja -kieleen. iOS-sovelluskehitys oli itselleni täysin ennenkokematonta ja uutta, eli opin samalla kun, työstin projektia eteenpäin. Ohjelmointiympäristönä oli xcode 14 ja ohjelmointikielenä toimi swift 5.8. Sovelluksen toimivuutta kokeiltiin iOS-puhelimella. Sovelluksen rakennus käynnistyi laajalti sovelluksen graafisen käyttöliittymän työstämisellä. Seuraavaksi perehdyin bluetooth-laitteiden ja näiden yhteyksien ohjelmointiin ja toiminnallisuuteen, sillä näistäkään minulla ei ollut lainkaan aikaisempa kokemusta. Tässä työssä hyödynnettiin Applen omaa ohjelmistokehystä ’CoreBluetooth’. Tällä kehyksellä voidaan hallinnoida ja luoda yhteyksiä bluetooth low energy -laitteisiin, kuten tässä projektissa käytetty elm327 obd2 -adapteri. Bluetooth palveluiden -kutsut ja palautuvan datan muoto aiheutti hieman hankaluuksia aluksi, mutta opin ja sain lopulta sen toimimaan halutulla tavalla.

Työ onnistui kohtuullisen hyvin. Toki applikaation graafinen käyttöliittymä jäi hivenen yksinkertaiseksi ja toiminallisuus osittain vajaaksi siitä mitä, olin projektin alussa kaavaillut, mutta päätavoite ja toiminallisuus toteutuivat, ja jatkokehitykselle jäi tilaa. Sovelluksen hyöty ja tarkoitus muodostuu siitä kuinka, sovellus tarjoaa yksinkertaisella käyttöliittymällä varustetun sovelluksen henkilöille, jotka ovat kiinnostuneita henkilö-autojensa obd2-järjestelmistä.

Abstract

Author:	Joel Haapaniemi
Title:	OBD2 Diagnostics Swift Application
Number of Pages:	31 pages
Date:	4.9.2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and communication technologies
Professional Major:	Software Development
Supervisors:	Keijo Länsikunnas, Senior Lecturer

Keywords: elm327, obd2

The objective of the thesis project was to create an application for the iOS-environment mobile phones. The essence of this application is an application that connects to an external elm327 obd2 Bluetooth adapter, which uses a wireless Bluetooth connection to communicate with the vehicle's obd system. This aforementioned adapter needs to be connected to the vehicle's obd2-port for this application to function as intended. The aim was to release the finished app on Apple's AppStore.

The project started off by learning about iOS development and getting acquainted with the development environment. The programming environment used was xcode 14 and the programming language swift 5.8. The application running and testing device was a iPhone 11 pro max mobile phone. The project took off by working on the graphical user interface. This was followed by getting acquainted with Bluetooth devices, their connectivity and how all this was to be programmed. In the project, Apple's own framework CoreBluetooth was used, this framework enables managing and creating connections to Bluetooth low energy devices, such as the elm327 obd2-adapter used here. There were some initial complications with the Bluetooth calls and data structures, but in the end it was got to work as intended.

All in all, the project met the goals set at the start of it. Admittedly the graphical user interface remained somewhat simple and plain, but the main goal and functionality of the project was achieved offering a simple obd2 application for anyone interested in their vehicles.

Sisällys

Lyhenteet

ODB: On-Board Diagnostics. Auton tietokoneen sisäinen järjestelmä, joka seuraa muun muassa moottorin ja voimansiirron järjestelmiä mahdollisten vikojen kannalta.

ELM327: Ohjelmoitu mikrokontrolleri, joka on suunniteltu keskustelemaan OBD-protokollan kanssa.

JOBD: Japanese On-Board Diagnostics. Auton tietokoneen sisäinen järjestelmä, joka seuraa muun muassa moottorin ja voimansiirron järjestelmiä mahdollisten vikojen kannalta. Japanin oma standardi.

EOBD: European On-Board Diagnostics. Auton tietokoneen sisäinen järjestelmä, joka seuraa muun muassa moottorin ja voimansiirron järjestelmiä mahdollisten vikojen kannalta. Euroopan oma standardi on verrattavissa yhdysvaltain OBD-standardiin.

1 Johdanto

Tässä opinnäytetyössä käydään läpi obd2-teknologiaa, mikä se on ja miten se toimii, sekä sen eroavaisuuksia kuten muun muassa Japanin oma standardi jobd tai euroopan eobd. Tämän lisäksi opinnäytteen pääasiallisena työnä tulee olemaan Apple:n uudella ohjelmointikielellä Swift-kielellä ohjelmoitu sovellus, joka keskustelee obd2-portilla varustetun ajoneuvon kanssa bluetooth-yhteyden ylitse ja muun muassa esittelee sovelluksen graafisessa käyttöliittymässä ajoneuvon reaali-aikaista statistiikkaa kuten lämpösensorien arvoja ja bensankulutusta.

Työssä käytetään ohjelmoinnin lisäksi iOS-puhelinta ja geneeristä elm327 obd2 bluetooth -adapteria. Toisin sanoen sovelluksen toimivuus vaatii loppukäyttäjältä vastaavanlaisen adapterin, joka tulee olla liitettynä oman autonsa obd2-portissa sovelluksen käytön aikana.

Työn insipiraationa oli oma kiinnostus autoja kohtaan, ja halusin luoda kyseisenlaisen sovelluksen omaan käyttöön. Olen myös halunnut kokeilla Applen sovelluskehitystä, joten tässä projektissa pääsin oppimaan paljon uutta ja luomaan käytännöllisen sovelluksen.

Työllä ei ole varsinaista toimeksiantoa tai asiakasta, mutta ideana oli julkaista valmis sovellus Applen App Storeen. Tarkoituksena oli luoda käytännöllinen applikaatio iOS-käyttöjärjestelmän käyttäjille. Jonkin verran tämänlaisia applikaatioita löytyy jo markkinoilta, mutta halusin luoda käyttäjäystävällisen ja yksinkertaisen bluetooth obd2 -applikaation. Uskoisin, että tälläiselle sovellukselle olisi kysyntää ja tarvetta.

2 OBD2-teknologia

OBD2 on järjestelmä, joka kehitettiin helpottamaan autojen ylläpitoa ja virhetilanteiden diagnosointia varten. Nykyajan autoissa on tietokoneet (ECU), toisella nimellä tunnettuna moottorin ohjausyksikkö. OBD2 on järjestelmä, joka pyörii auton tietokoneella. Se jatkuvasti seuraa lukuisia auton sensoreita tarkistaen, että kaikki arvot ovat tasolla joiden, niiden kuuluisi olla. Mikäli jokin sensori syöttää arvoja, jotka eivät ole suositellun rajan sisällä, lähettää obd2-järjestelmä tästä vikakoodin auton tietokoneelle, joka taas sytyttää moottorin vikavalon tai moderneimmissa autoissa näyttää vikakoodin suoraan kojelaudan näytöllä.

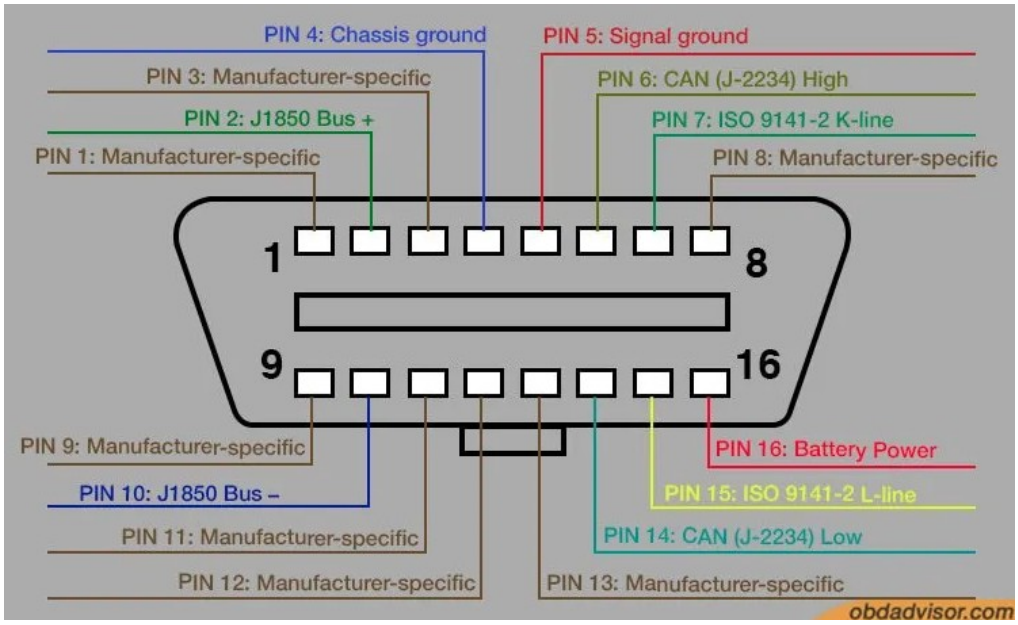
OBD-järjestelmä alunperin myös kehitettiin helpottamaan autojen päästömittauksia. Näin voidaan varmistua, että ensinnäkin auton moottori toimii niin kuin sen tulisi ja, se, että auto ei saastuta ympäristöä liiallisilla päästöillä.

OBD2 on standardisoitu malli ja, itse fyysinen portti on aina 16-pinninen, mutta kuitenkin on käytössä viisi erilaista protokollaa eri autonvalmistajien kesken. Tämä portti tunnetaan nimellä SAE J1962 [1]. Lyhyesti sanottuna protokollien välillä pinnien taajuus ja signaalit eroavat hieman toisistaan. Yleisin käytetty protokolla on; ISO 9141-2. OBD2-porttia tuli myös kahta eri variaatiota OBD2-A ja OBD2-B. Näistä OBD2-A on 12V:n autoihin eli se yleisempi käytetty henkilöautoissa, OBD2-B on 24V:n ajoneuvoihin eli raskaammalle kalustolle kuten rekoille ja kuorma-autoille.

OBD2-portin kautta voidaan lukea ajoneuvon vikakoodit. Nämä vikakoodit ovat standardisoituja viiden kirjaimen koodeja. Vikakoodit alkavat aina kirjaimella, joita vaihtoehtoina on P, B, C ja U. P-kirjain merkitsee, että vikakoodi kohdistuu ajoneuvon voimansiirtoon, B-kirjain on tarkoitettu ajoneuvon korille, C-kirjain tarkoittaa ajoneuvon alustaa ja U-kirjain on ajoneuvon sähköjärjestelmä. Tämän ensimmäisen kohdistavan kirjaimen jälkeen tulee neljä numeroa ja kirjainta, mikä yksilöllistää tarkasti ajoneuvon viat.

2.1 JOBD

JOBD on Japanissa käytetty muunneltu standardi OBD2-järjestelmästä. Tämä järjestelmä eroaa muun muassa päästöjen mittauksissa, jotka ovat Japanissa tarkempia kuin muualla maailmassa. Kuitenkin JOBD on hyvin samankaltainen kuiten OBD2-protokolla, ja järjestelmä käyttää samaa 16-pinnistä SAE J1962 -porttia.

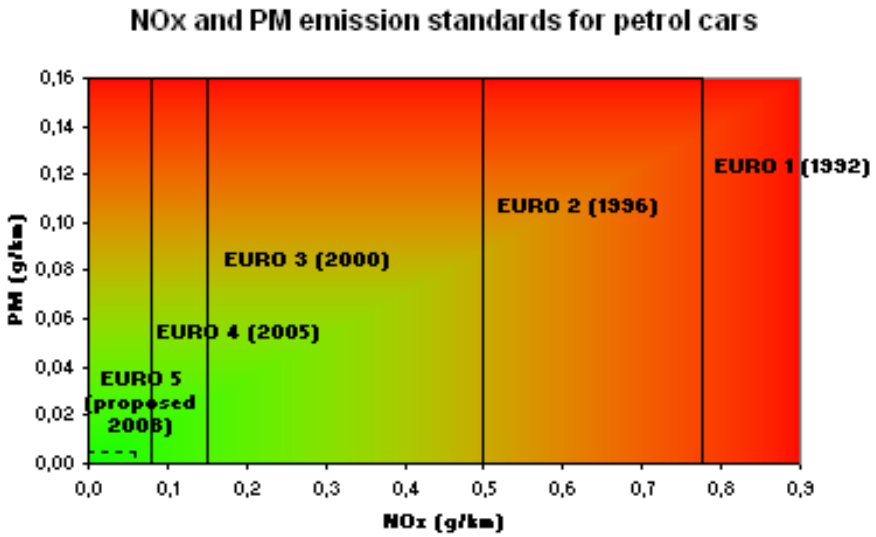


Kuva 1: OBD2-portin yksittäiset pinnit

Pääasialliset erot ovat päästöjen mittaus ja vikakoodien muoto, luku ja nollaus. Yksinomaan Japanin markkinoille valmistetut autot ovat täten jobd-standardin mukaisia eivätkä obd2 kuten muissa maissa hyvin yleisesti. Poikkeuksena ovat autot, joita valmistetaan myös ulkomaille vietäväksi. Näistä löytyy perinteinen obd2-standardin mukainen portti. Tämän vuoksi muissa maissa laki vaatii modernin auton, joka varustetaan obd2-protokollaa tukevalla portilla. Japanin-maalla obd2-protokollan tuettavuus ei ole lain vaatimaa.

2.2 EOBD

EODB on Euroopan versio OBD2-järjestelmästä. EODB käyttää samaa 16-pinnistä SAE J1962 -diagnostiikkaporttia kuin OBD2. Eroavaisuus näiden kesken on ajoneuvojen päästöihin liittyvät järjestelmät. Tämä johtuu Euroopan sovituista EURO 5- ja 6 -päästöstandardeista. EOBD-järjestelmää on myös saatavilla EOBD2-järjestelmää, mutta tämä ei kuitenkaan ole sama kuin OBD1 ja OBD2, vaan EOBD2 on vain hieman erikoitustunut versio EOBD-järjestelmään verrattuna, eli lyhyesti sanottuna se on vain markkinointitermi eikä uusi standardi.



Kuva 2: EURO-päästöluokitukset

2.3 ELM327

ELM327 on ohjelmoitu mikrokontrolleri, jolla voidaan SAE J1962 -portin kautta keskustella OBD2-järjestelmän kanssa. Tämän mikrokontrollerin käyttämä komentoprotokolla on yleisin käytetty protokolla obd-järjestelmien kanssa keskusteluun [2].

ELM327 AT Commands

Version in which the command first appeared…

version	Command	Description	Group
1.0	@1	display the device description	General
1.3	@2	display the device identifier	General
1.3	@3 ccccccccccc	store the device identifier	General
1.0	<CR>	repeat the last command	General
1.0	AL	Allow Long (>7 byte) messages	OBD
1.2	AR	Automatic Receive	OBD
1.2	AT0	Adaptive Timing Off	OBD
1.2	AT1	Adaptive Timing Auto1	OBD
1.2	AT2	Adaptive Timing Auto2	OBD
1.0	BD	perform a Buffer Dump	OBD
1.0	BI	Bypass the Initialization sequence	OBD

Kuva 3: ELM327 AT -komentoja

ELM327-kontrollerin kanssa voidaan keskustella muun muassa Wi-Fi, Bluetooth ja USB-yhteyksien välitse. ELM327-komentofunktioita on muun muassa vikakoodien luku, ajoneuvon nopeus ja moottorin kierrosnopeus.

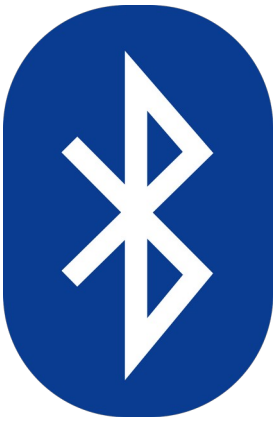
Alkuperäinen ELM327-mikrokontrolleri on ELM Electronics -yhtiön kehittämä ja valmistama. Kyseinen yhtiö ei enää kuitenkaan ole toiminnassa, joten alkuperäisia elm327-mikrokontrolleja ei enää valmisteta vaan nykyään ovat muut yhtiöt kopioinneet mikrokontrollerin ja valmistavat tätä alkuperäistä mallintavaa piirisarjaa.

Yhtiön yhä ollessa pystyssä piirisarjan kopiointi kävi erittäin helposti, sillä ELM Electronics ei ollut pannut minkäänlaista koodin suojausta heidän mikrokontrollerilleen, joten kaiken binäärikoodin pystyi kopioimaan alkuperäiseltä kontrollerilta toiselle kopiokontrollerille.

Elm327-mikrokontrollerin kanssa keskusteluun käytetty komentosetti on hyvin samankaltainen kuin Hayes AT -komennot. Hayes AT -komennot olivat Dennis Hayesin kehittämät komennot vuonna 1981, joita käytettiin verkkomodeemien kanssa keskusteluun. Komennot ovat tekstipohjaisia standardoituja komentoja joilla, mikrokontrollerin kanssa voidaan keskustella [3.].

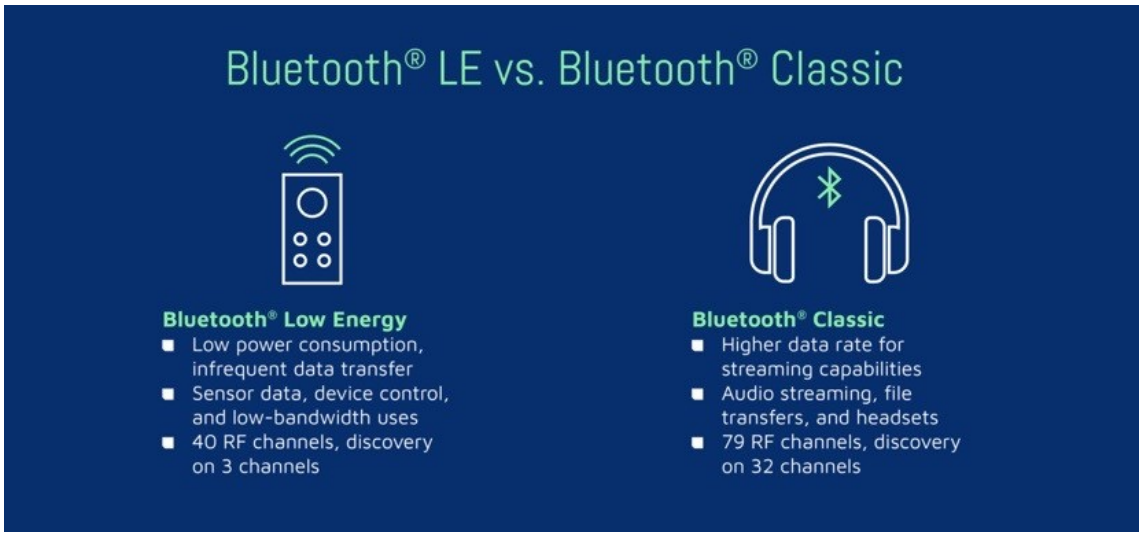
3 Bluetooth

Bluetooth on lyhyen kantaman langaton teknologia, jolla voidaan siirtää dataa laitteiden kesken. Bluetooth toimii 2.4Ghz:n radiotaajuuden tienoilla ja, sen kantama on keskimäärin 10 metriä.Bluetooth nimi- tuli ehdotuksena vuonna 1997 Intelin työntekijältä Jim Kardachilta. Nimi tuli 900-luvun aikana eläneestä tanskalaiskuninkaasta Harald Bluetooth. Tarkoitus oli vain olla projektinimi, mutta aikataulun kiristyessä bluetoothin julkaisuun tämä jäi teknologian viralliseksi nimeksi. Myös bluetoothin logo vie juurensa 900-luvulle. Se on yhdistetty riimukirjain jossa, on riimu Hagall ja Bjarkan, kuningas Haraldin nimikirjaimet.



Kuva 4: Bluetooth-logo

Bluetooth-dataa siirretään paketeissa 79 kanavan yli, jokaisen ja kanavan leveys on 1 Mhz. Bluetooth Low Energy tai lyhyesti BLE käyttää 40 kanavaa joista, jokainen kanavan leveys on 2 Mhz [4.]. Bluetooth käyttää muun muassa samaa taajuutta kuin Wi-Fi eli 2,4 Ghz. Tämä tarkoittaa sitä, että jos läheisyydessä on sekä bluetooth- että Wi-Fi-laitteita yhteyksissä, voivat datapaketit korruptoitua tai mennä ristiin. Tämän ongelman ratkaisemiseksi kehitettiin ns. taajushyppely, jossa datapaketteja lähetetään monen kanavan – eli taajuuden – kautta. Näin vältetään datan korruptoitumista ja, tämän lisäksi vielä taajuushyppely on tilanteeseen muokkautuva eli jos jotkin kanavat ovat ruuhkaisia, protokolla vaihtaa datapakettien lähetykset toisille kanaville. Tämä on ns. adaptiviinen taajushyppely [5.].



Kuva 5: BLE vs Bluetooth

Bluetooth-pakettiprotokolla käyttää keskitin/laitearkkitehtuuria, jossa keskitin voi keskustella yhtäaikaisesti seitsemän laitteen kanssa. Keskitin lähettää dataa tiettyjä kanavia pitkin ja, laite ottaa vastaa datan määrittyjä kanavia pitkin. Näin keskitin ja laite voivat jatkuvasti keskustella toisilleen ilman datakatkoksia tai viiveitä.

Bluetoothilla voidaan eliminoida turhat fyysiset kaapelit lähietäisyydellä olevien laitteiden välillä. Sen lisäksi se on standardoitu protokolla eli useimmat bluetoothilla varustetut laitteet voivat kommunikoida keskenään, toisin kuin fyysiset kaapelit, joita löytyy montaa eri sorttia. Nykyisin bluetooth on varsinkin yleisessä käytössä oleva teknologia, koska se on halpa, käytännöllinen ja tärkeimpänä toimiva tapa laitteiden väliseen keskusteluun. Tämänhetkinen uusin versio bluetoothista on versio 5.3.

Bluetooth ei suinkaan ole ainut langaton teknologia datan siirtoon. Toinen hyvin vastaavanlainen teknologia on Wi-Fi, mutta tämän teknologian toteuttaminen on kalliimpaa kuin bluetooth. Wi-Fi myös käyttää enemmän sähkövirtaa, joten moneen käyttötarkoitukseen se ei olisi kovinkaan järkevä sovellus datan siirtoon tai kommunikointiin laitteiden välillä.

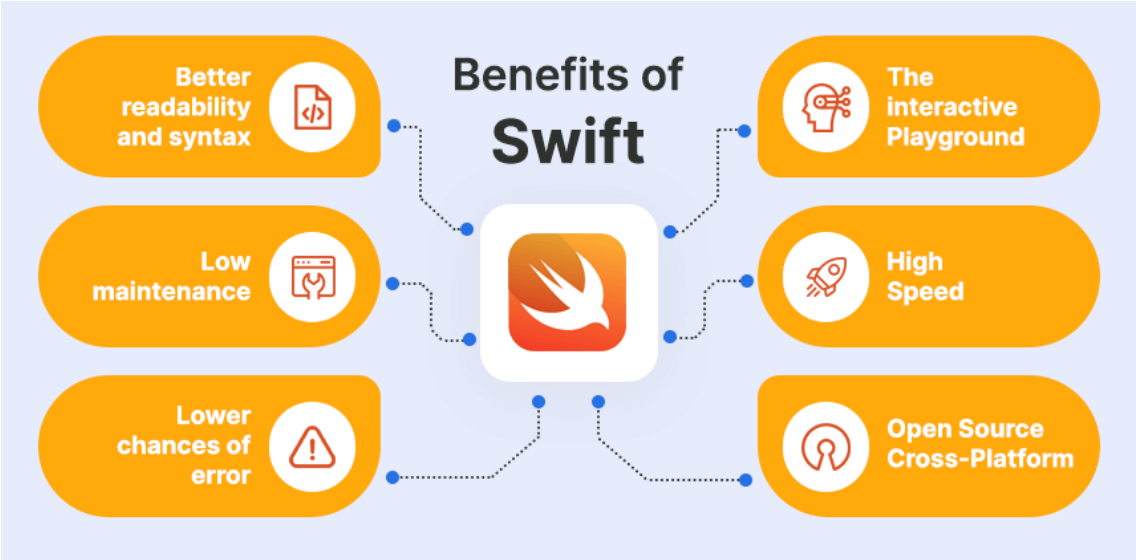
4 Työkalut

4.1 Swift

Swift on Applen kehittämä objektiohjelmointikieli, joka tuli aiemmin käytetyn Objective C:n tilalle. Periaatteessa Swift on samankaltainen kuin Objective C, mutta suoraviivaisempi ja mukaistettavissa muihin objektiorientoitu ohjelmointikieliin kuten Javaan tai C#:aan. Swift julkaistiin ensimmäistä kertaa vuonna 2014 ja on tänä päivänä tullut standardiksi ohjelmointikieleksi iOS/MacOS -sovelluksille. Uusin versio tällä hetkellä on Swift 5.7.1.

Swift verrattuna esimerkiksi C++/C#:aan on nopeampi, turvallisempi ja helpompi ohjelmointikieli. Swiftillä on hankalempaa kirjoittaa huonoa tai epäturvallista koodia. Koodi ei ensinnäkään mene läpi ellei koodia ole kirjoitettu oikea-oppisesti. Swift ei myöskään ole lukittu Applen omaan ympäristöön, vaan se on avoin ja maksutun kieli, joka on käytettävissä Windows- ja Linux-ympäristöissä.

Swift tukee muun muassa monikkoarvojen palautusta funktioilla. Swift-ohjelmoinilla voidaan käydä läpi datasettiä kuten taulukkoa paljon nopeammin ja helpommin kuin muilla ohjelmointikielillä. Myös virhetilanteiden käsittely Swiftillä on paljon kehittyneempi verrattuna muihin ohjelmointikieliin. Swiftillä on ajonaikaisia virhetilanteiden käsittelyyn muista kielistä tuttu ’throw’ ja ’catch’in lisäksi ’propagate’ ja ’manipulate’.



Kuva 6: Swift ohjelmointikielen -hyödyt

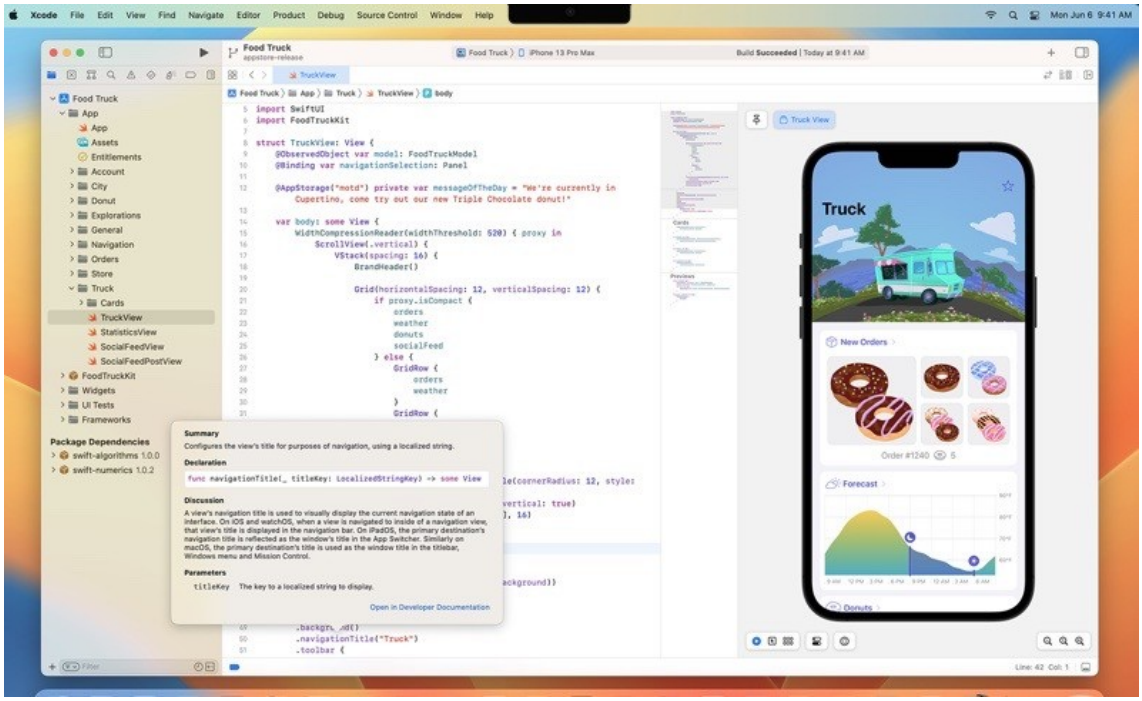
Swift käyttää kehittyneempää koodin ajonhallintaa, joilla voidaan muun muassa ehdoilla vaikuttaa siihen miten, koodia ajetaan tai miten se käyttäytyy. Toki tämä on mahdollista muissakin ohjelmointikielissä, mutta Swiftissä tämä on tehty selkeämmäksi ja helpommaksi kehittäjälle.

Tämän lisäksi Swift käyttää ARC-järjestelmää eli ’Advanced Reference Countingia’. Tämä järjestelmä ylläpitää muistinhallintaa. Ohjelmakoodin ajon aikana ARC ylläpitää kirjaa esimerkiksi luokkaolioista ja vapauttaa sen myöhemmin muistista mikäli, oliosta muodostuu käyttämätön [6.].

Swift-luokkajärjestelmä tunnetaan nimellä ’Struct’ ja, se on paljon joustavampi kuin tyyppilliset luokat muissa ohjelmointikielissä. Swiftissä voidaan luokkakutsuissa laajentaa sen toiminallisuutta käyttämällä laajennuksia tai ylimääräisiä metodeja. Tämän lisäksi Swiftistä löytyvät perinteiset ’Class’-luokat.

4.2 Xcode

Xcode on Applen kehittämä ohjelmointiympäristö, jolla voidaan kehittää sovelluksia iOS- ja MacOS-käyttöympäristöille. Xcode tukee muun muassa kieliä kuten Swift, C, Java ja Python [7]. Xcode julkaistiin vuonna 2003 ja, sen tämänhetkinen uusin vakaa versio on Xcode 14. Xcode käyttää Clang-kääntäjää koodin kääntämiseen. Xcode tukee git-versionhallintaa ja siinä on graafisen käyttöjärjestelmän rakentaja, jolla voidaan helposti rakentaa graafista käyttöjärjestelmää.



Kuva 7: Xcode 14 -ohjelmointiympäristö

Xcoden mukana tulee Playgrounds-ympäristö, jossa voidaan nopeasti ja helposti kokeilla koodia ja samalla nähdä miten, koodi käyttäytyy ja miltä sovelluksen graafinen käyttöliittymä näyttää. Tämä on erittäin hyödyllinen työkalu uusille Swift-kehittäjille, jotka vielä oppivat kieltä. Tällä myös onnistuu prototyyppikoodin kokeilu helposti ja vaivattomasti.

Hieman kritiikkiä voisi kuitenkin antaa tälle ohjelmointiympäristölle siitä kuinka, sen käyttö vaatii lähes uusinta versiota macOS-käyttöjärjestelmästä, jotta sitä voisi käyttää kehitykseen. Tällä rajoitetaan turhaan mahdollisia sovelluskehittäjiä joilla, ei ole tarpeeksi modernia macOS-käyttölaitetta. Tosin sanottakoon, että kolmannen osapuolen avulla on Xcode-sovelluskehitys myös mahdollista vanhemmille ei virallisesti -tuetuilla laitteilla.

5 OBD2 Diagnostiikkasovellus ja sen kehitys

Tässä projektissa halusin rakentaa iOS-sovelluksen, joka pystyisi kommunikoimaan ELM327-mikrokontrollerin kanssa ja tätä kautta saisin sovelluksen, jossa voisin esittää käyttäjälle OBD2-vikakoodeja ja reaali-aikaisia auton sensorien arvoja ja dataa.

Työ lähti käyntiin sillä, että rupesin tutustumaan iOS-kehitykseen Applen Swift-ohjelmointikielellä ja heidän ohjelmointiympäristöönsä Xcode. Myös Bluetooth-teknologia oli itselleni täysin uutta, joten opin tässä samalla kun, työstin projektia eteenpäin.

Alkujaan hankaluuksia tuotti uusi ohjelmointikieli ja bluetooth-yhteydet sekä näiden laitteiden ominaiset komennot. Bluetooth-laitteiden kanssa keskusteluun käytin Applen kehittämää sovelluskehystä Core Bluetooth, jolla voidaan kirjoittaa koodia keskustelemaan bluetooth low energy- ja bluetooth classic -laitteiden kanssa. Tämä sovelluskehys ja sen käyttö on hyvin dokumentoitu ja esitetty Applen omilla dokumentaatiosivuilla, mutta ongelmaa aiheutti se kuinka, bluetooth low energy- ja bluetooth classic -laitteiden kanssa palvelukutsuista ei löytynyt minkäänlaista dokumentaatiota.

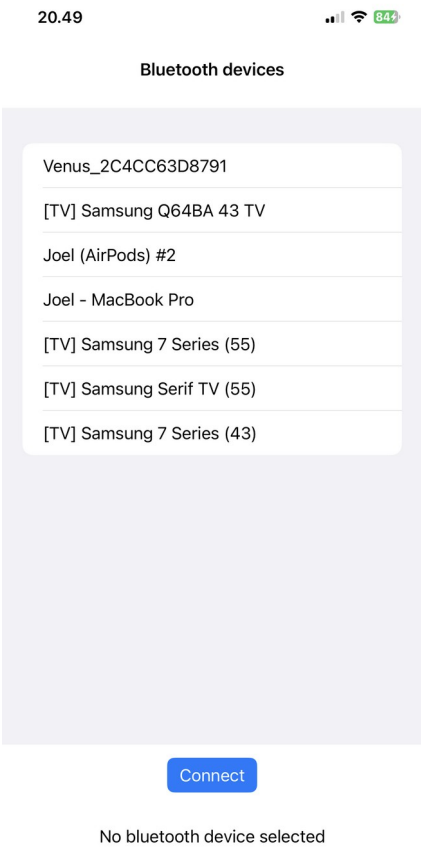
Swift-ohjelmointikielellä ohjelmointi oli hyvinkin luontevaa ja helppoa, jos on aikaisempaa kokemusta muista olio-ohjelmointikeskittyneistä kielistä kuten Javasta tai C++:sta. Paljon myös löytyy ohjelmointitutoriaaleja ja muutakin kehittäjien dokumentaatiota siitä kuinka, Swiftillä tulisi ohjelmoida tai ratkaista jokin ongelmatilanne.

Tätä projektia lähestyin ajatuksella, että yksinkertaisuus on valttia. Normaalista mvc-käytännöstä poiketen kaikki ohjelmakoodi on yhdellä Swift-moduulilla tai luokalla. Jatkokehityksen kannalta tämänkaltainen ohjelmointi-ideologia ei välttämättä ole paras, mutta kun ottaa kuitenkin huomioon projektin koon ja ohjelmoijan kokemuksen Swift-ohjelmoinnista. Uskon tämän lähestymistavan olevan ideaali.

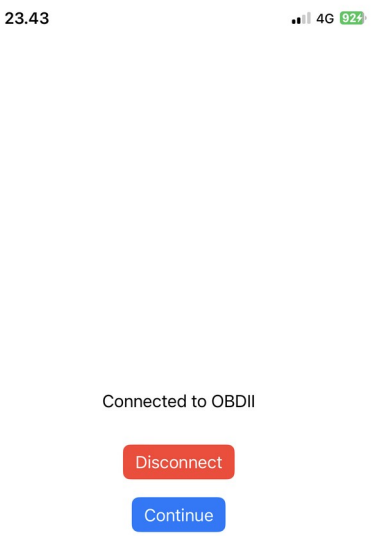
```
class BluetoothViewModel: NSObject, ObservableObject {
    private var centralManager: CBCentralManager?
    private var peripherals: [CBPeripheral] = []
    private var placeholderPeripheral: CBPeripheral?
    private var connectedPeripheral: CBPeripheral?

    @Published var peripheralNames: [String] = []
    @Published var connected: Bool = false
    @Published var loadMainView: Bool = false
    @Published var connectedPeripheralName: String?

    // Characteristics UUIDs
    let modelNumberString = CBUUID(string: "2A24")
    let manufacturerNameString = CBUUID(string: "2A29")
    let elmCodeNotify = CBUUID(string: "AE02")
    let elmCodeRead = CBUUID(string: "AE10")

    override init() {
        super.init()
        self.centralManager = CBCentralManager(delegate: self, queue: .main)
   }
}

extension BluetoothViewModel: CBCentralManagerDelegate, CBPeripheralDelegate {

    private func ModelNumberString(from characteristic: CBCharacteristic) -> String {
      guard let characteristicData = characteristic.value,
        let byte = characteristicData.first else { return "Error" }
```

Kuva 8: BluetoothViewModel-olio

Ohjelmakoodin pääosassa onkin bluetooth-yhteyksiä hallinnoiva olio, joka ylläpitää bluetooth-laitteiden yhteyksiä ja laitteiden välillä kulkevaa dataa. Tämä data käännetään tekstiksi, joka päivitetään näkyväksi sovelluksen graafiselle käyttöliittymälle. Tämä olio käyttää hyväkseen Applen kehittämä Core Bluetooth -ohjelmistokehys tuo mukanaan hyödyllisiä funktioita, joilla voidaan hallinnoida bluetooth laitteiden yhteyksien tilaa ja datakutsuja.



Kuva 9: Sovelluksen alkunäyttö



Kuva 10: Yhteyden vahvistusnäyttö

Ohjelmakoodin ajo ja sen käyttö alkaa yhdistämällä bluetooth-laitteeseen. Ohjelma sallii yhdistämisen mihin tahansa läheisyydessä olevaan BLE-laitteeseen, mutta kuitenkaan mitään toiminallisuutta tähän ei tietenkään ole kehitetty kun, ideana on yhdistää iOS-päätelaite bluetooth obd2 -päätteeseen.

Bluetooth yhteyksistä sen verran enempi, että laitteet käyttävät rekisteröityjä ’Service UUID’, joita käytetään ohjelmointitarpeisiin. Nämä rekisteröidyt koodit löytyvät kansainvälisen bluetooth SIG -yhtiön sivuilta. Näitä tarvitaan ensinnäkin datakutsujen tekemiseen, mutta myös datan kääntämiseen/purkamiseen. Käyttämäni bluetooth obd2 -adapteri on kiinalaiskopio elm327-mikrokontrollerista (alkuperäisiä ei ole enää saatavilla) ja tämän piirisarjan UUID-koodeista ei ole minkäänlaista dokumentaatiota. Tämä hieman

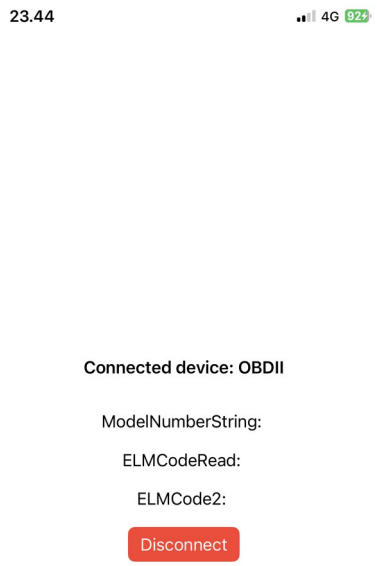
hankaloittaa asioita, mutta onneksi UUID ja/tai komennot ovat yleensä samankaltaisia eri laitteiden välillä, joten kokeilemalla voinee selvittää mikä, saisi datan kulkemaan laitteiden välillä.

Projektin alkuperäinen nimi oli ’jobd2’ ja sivuideana olikin hieman jobd-protokollan tukeminen, mutta puolessa välin työtä päätin, että ei ole kannattavaa käyttää tähän resursseja, varsinkin kun Japanin standardista ei ole lainkaan dokumentaatiota ainakaan englannin kielellä. Projektin uudeksi nimeksi tuli ’ezOBD’. Markkinoinnin kannalta uskon tämän nimen olevan sopivampi. Myös vaihdoin sovelluksen ikonin Japani-aiheinen ’Torii’-portista paljon yksinkertaisempaa ja eleganttisempaan tekoälyn kehittämään kuvakkeeseen, jota itse jälkeenpäin muokkasin kuvanmuokkaustyökalulla saadakseni värimaailman itseäni miellyttäväksi.

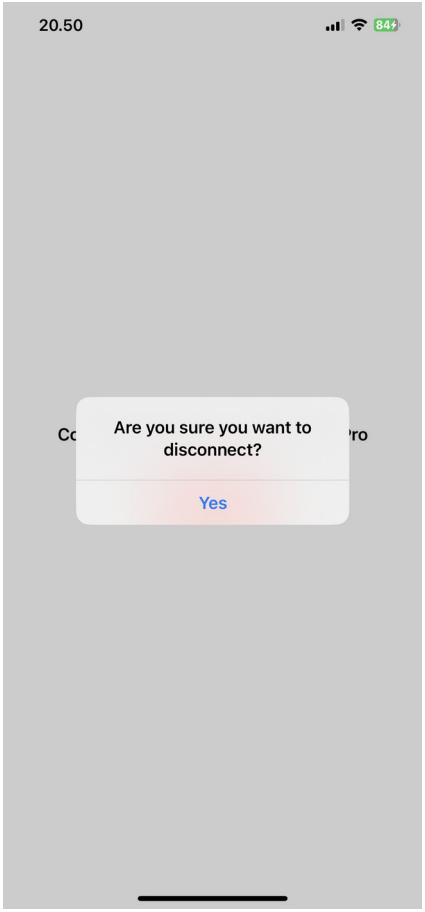
Kuva 11: Sovelluksen kuvake/ikoni

Sovelluksen käyttöliittymästä ja sen filosofiasta hieman enempi; käyttöliittymä on suunnitelty käyttäjälähtöisesti niin, että sen käyttö on mahdollisimman

luontevaa, selkeää ja erityisesti helppoa. Käyttöliittymän tärkeys on todella tärkeä seikka johon, kannattaakin käyttää suunnitteluresursseja, sillä jos käyttöliittymä ei ole hyvä ja luonteva, ei sovelluksen ominaisuuksilla tai funktioilla ole lainkaan merkitystä, jos applikaation käyttö ei ole mieluisaa loppukäyttäjälle. Sovelluksen käyttöliittymä sekä ohjelmakoodi on kirjoitettu englannin kielellä.



Kuva 12: Sovelluksen yhdistynyt-tila



Kuva 13: Sovelluksen yhteyden katkaisun varmennusnäyttö

Sovelluksen käyttöliittymä on hyvinkin yksinkertainen ja selkeä kuten nähdään kuvista 12 ja 13, mutta kuten aiemmin mainitsin yksinkertaisuus ,on valttia, varsinkin käyttöliittymissä. Kovin monet sovellukset ja/tai verkkosivut koittavat

ylittää taiteellisia näkemyksiä ja, tämä tulee käytettävyyden heikkenemisellä. Käytettävyys on aina tärkeämpi kuin graafiset taiteelliset visiot.

Koodin ajo kulkeutuu hyvin vahvasti tuon bluetoothViewModel-olion ohjaamana. Tämä olio on laajennus tuosta corebluetoothin laitemanagerista. Lyhyesti sanottuna, sovelluksen ajon alussa tämä manageri etsii lähellä olevat bluetooth-laitteet. Kun johonkin laitteeseen yhdistetään, tämä manageri hakee laitteen tarjoamat palvelut ja tiedot. Näitä kutsuja voidaan sitten myöhemmin käyttää datan ulossaamiseksi. Itse datan muoto on enimmäkseen ’string’-tyyppiä, joka parsitaan halutun kaltaiseksi, jonka voi sitten esittää käyttäjälle järkevästi sovelluksen käyttöliittymässä.

```
func peripheral(_ peripheral: CBPeripheral, didDiscoverServices error: Error?) {
    print("The services of the connected device: \(peripheral.services?.description ?? "no services available")")
    discoverCharacteristics(peripheral: peripheral)
}

func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService,
                error: Error?) {
  guard let characteristics = service.characteristics else { return }

  for characteristic in characteristics {
      print(characteristic)
      if characteristic.properties.contains(.read) {
        print("\(characteristic.uuid): properties contains .read")
        peripheral.readValue(for: characteristic)
      }
      if characteristic.properties.contains(.notify) {
        print("\(characteristic.uuid): properties contains .notify")
        peripheral.setNotifyValue(true, for: characteristic)
      }
  }
}
```

Kuva 14: Laitteiden palveluiden löytämisen koodi

Pienen hämmenyksen ja päävaivan aiheutti se kuinka, bluetooth low energy -laitteet yleisesti kommunikoivat unicode-datamuodossa. Jotta tämä data olisi käytettävissä ascii-muodossa, se tulee ensin kääntää sille.

```
private func ModelNumberString(from characteristic: CBCharacteristic) -> String {
    let unicodeString = String(data: characteristic.value!, encoding: String.Encoding.utf8) ?? "n/a"
    modelNumberString = unicodeString
    return unicodeString
}

private func elmCodeRead(from characteristic: CBCharacteristic) -> String {
    let unicodeString = String(data: characteristic.value!, encoding: String.Encoding.utf8) ?? "n/a"
    elmCodeRead = unicodeString
    return unicodeString
}

private func elmCode2(from characteristic: CBCharacteristic) -> String {
    let unicodeString = String(data: characteristic.value!, encoding: String.Encoding.utf8) ?? "n/a"
    elmCode2 = unicodeString
    return unicodeString
}
```

Kuva 15: BLE-datan käännös utf8 string -muotoon

Projektin loppua kohden tuli muutoksia kehitysprosessiin niin, että siirryttiin reaalimaailman testauksesta simuloituun testaukseen. Kehitys- ja testausprosessi muuttui tässä niin, että siirryttiin aiemmasta menetelmästä, joka oli, että elm327 obd2-adapteri oli kiinni oikeassa henkilöauton obd2-portissa. Tässä menetelmässä oli toki se hyöty, että voitiin heti todeta omin silmin se, että sovellus toimii kuten sen kuuluisi, mutta toki se oli hiukan hankalaa tehdä lennosta kehitystä henkilöauton ratin takana ja tyhjäkäyttää moottoria. Uudeksi menetelmäksi tuli simuloitu obd2-testiympäristö. Näin kehitysprosessi helpottui ja tuli sujuvammaksi.

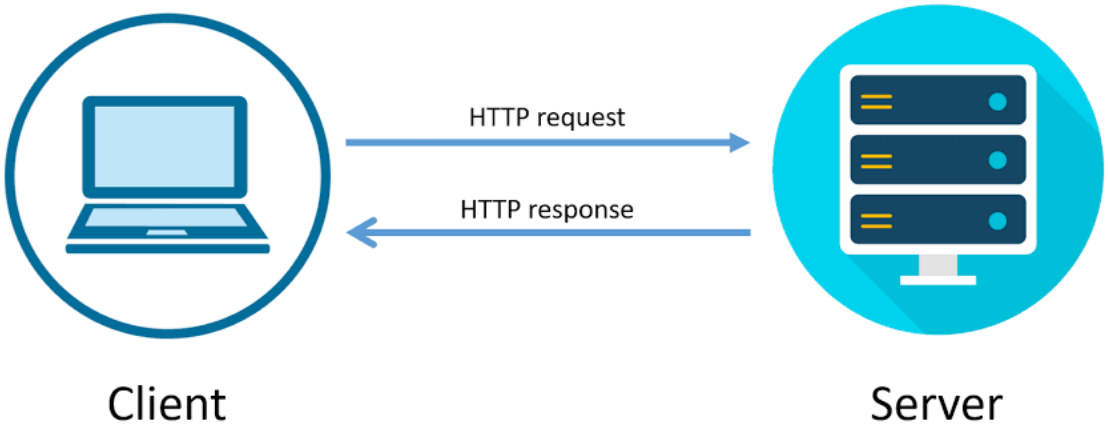
Tämä siirto sovelluksen kehityksessä kuitenkin aiheutti sen, että vaihdettiin tämän datayhteyden muoto bluetoothista http-protokollaan. Bluetooth-toiminallisuus kuitenkin jätettiin sovellukseen niin kuin olin sen alunperinkin kaavaillut, mutta nyt vain itse obd2-datakeskustelu laitteiden välillä tapahtuu http-protokollan kautta eikä bluetooth-protokollan ylitse.

Tcp-protokolla on tietoliikenneprotokolla, joka määrittelee tietoliikenneyhteyden säännöt eli miten dataa lähetetään ja vastaanotetaan. Http-pyynnöt taas ovat itse datarakennelmat tai paketit eli näillä pyynnöillä voidaan lähettää tai vastaanottaa dataa.

```
joelhaapaniemi@Joel-MacBook-Pro ~ % python3 -m elm -n 35000 -s car
2023-06-15 16:19:36,713 - root - INFO -

ELM327 OBD-II adapter emulator v3.0.0 started at TCP/IP network port 35000.

Emulator scenario switched to 'car'
Welcome to the ELM327 OBD-II adapter emulator.
ELM327-emulator is running on TCP network port 35000.
Type help or ? to list commands.
```

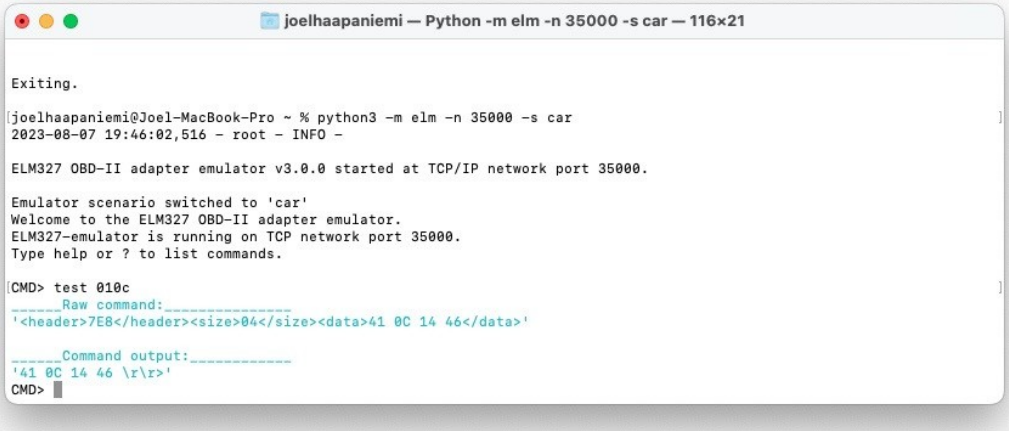
Kuva 16: Python obd2 elm327 -emulaattori

Tämä edellä mainittu simulaattori pyörii lokaalissa verkossa, johon puhelimella ajettava sovellus muodostaa yhteyden tcp-protokollan ylitse ja keskustelu laitteiden välillä toteutuu http-pyyntöjen avulla. Tämä Python-ohjelmointikielellä toteutettu emulaattori [8] käyttää samoja AT-komentoja kuin fyysinen elm327 obd2 bluetooth -adapteri, jota käytettiin tämän projektin alkuvaiheessa. Ainoana erona on vain tapa jolla, tämä data liikkuu näiden yhdistyneiden laitteiden välillä.



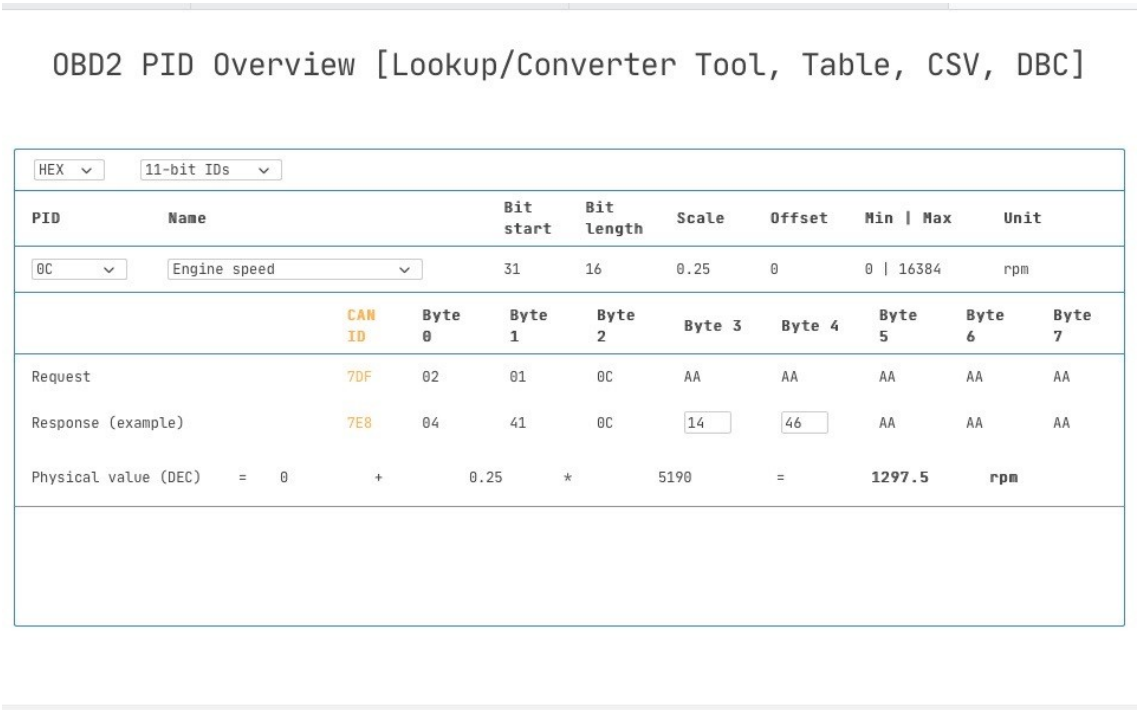
Kuva 17: HTTP-pyyntö

Aiemmin mainittu elm327-emulaattori tuotti hiukan hankaluuksia sovellukseni yhdentämisen kanssa, eritoten se kuinka tämän projektin kehittäjä ei juurikaan ollut dokumentoinnut tcp/ip-protokollan käyttämisestä yhteytenä, muutoin kuin että sen tulisi toimia. Yhteyden luominen kylläkin onnistuu, mutta vaikeuksia on itse http-pyyntöjen kanssa, kun ei ole dokumentaatiota tästä, että millaisia hakupyyntöjä tämä emulaattori odottaa, jää ainoaksi menetelmäksi edettäväksi vain sattumanvarainen kokeilu. Ajattelin siirtyä tästä kommunikaatiotavasta perinteiseen bluetooth-yhteyteen, siitä olikin myös paremmin dokumentaatiota, mutta ongelmana tuli se kuinka, vaaditut työkalut eivät ole tuettuina projektin kehitysalustalla (MacOS Ventura). Tästä voisin sivulla mainita kuinka, erittäin tärkeä on kattava dokumentaatio, sovellus tai työkalu voi olla vaikka kuinka toimiva ja hyvä, mutta on se lähes käyttökelvoton ilman kattavaa dokumentaatiota.



Kuva 18: OBD2 PID -komento testi Python elm327 -emulaattorilla

Itse elm327-emulaattori toimii muutoin halutulla tavalla, pystyn ainakin testailemaan OBD2 PID-komentoja ja niiden vastauksia tältä simuloidulta obd2-järjestelmältä. Vastaukset ovat 11-bittisiä heksadesimaaleja, joiden kääntämiseen löysin hyvän työkalun.



OBD2 PID Overview [Lookup/Converter Tool, Table, CSV, DBC]

HEX | 11-bit IDs

PID	Name	Bit start	Bit length	Scale	Offset	Min | Max	Unit
0C	Engine speed	31	16	0.25	0	0 | 16384	rpm

	CAN ID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Request	7DF	02	01	0C	AA	AA	AA	AA	AA
Response (example)	7E8	04	41	0C	14	46	AA	AA	AA

Physical value (DEC) = 0 + 0.25 * 5190 = **1297.5** **rpm**

Kuva 19: OBD2 PID-komentojen vastauksien käännöstyökalu

OBD2 PID -kommennot toimivat niin, että eri datan pyyntimuotoja on kymmenen. Voidaan esimerkiksi pyytää jonkun sensorin tämänhetkinen reaalidata tai sitten vaikka säilöttyä vanhempaa dataa obd-järjestelmät kuten muun muassa vikakoodit. Datapyynnön muoto on niin, että kaksi ensimmäistä kirjainta määrittelee pyyntimuodon ja kaksi jälkimmäistä kirjainta määrittelee halutun datan. Kuten kuvassa 18, lähetämme ’010C’-pid komennon, ’01’ määrittelee, että pyydetään tämänhetkistä reaali-aikaista dataa ja ’0C’ on standardoisun pid-taulukon mukaan ’moottorin kierrosnopeus’.

Mode(hex)	PID(hex)	Data bytes returned	Description	Min value	Max value	Units	Formula
01	00	4	PIDs supported [01 - 20]				Bit encoded [A7..D0] == [PID 0x01..PID 0x20]
01	01	4	Monitor status since DTCs cleared. (Includes malfunction indicator lamp (MIL) status and number of DTCs.)				Bit encoded. See below.
01	02	8	Freeze DTC				
01	03	2	Fuel system status				Bit encoded. See below.
01	04	1	Calculated engine load value	0	100	%	A*100/255
01	05	1	Engine coolant temperature	-40	215	°C	A-40
01	06	1	Short term fuel % trim—Bank 1	-100 (Rich)	99.22 (Lean)	%	(A-128) * 100/128
01	07	1	Long term fuel % trim—Bank 1	-100 (Rich)	99.22 (Lean)	%	(A-128) * 100/128
01	08	1	Short term fuel % trim—Bank 2	-100 (Rich)	99.22 (Lean)	%	(A-128) * 100/128
01	09	1	Long term fuel % trim—Bank 2	-100 (Rich)	99.22 (Lean)	%	(A-128) * 100/128
01	0A	1	Fuel pressure	0	765	kPa (gauge)	A*3
01	0B	1	Intake manifold absolute pressure	0	255	kPa (absolute)	A
01	0C	2	Engine RPM	0	16,383.75	rpm	((A*256)+B)/4
01	0D	1	Vehicle speed	0	255	km/h	A
						° relative	

Kuva 20: OBD2 PID -taulukko

Jos katsotaan kuvan 20 taulukkoa, voidaan todeta, että yhtälö obd2 pid -komennon ’010C’ käännös desimaaliluvuksi on ((256*A)+B)/4, jossa A on 14 heksadesimaali eli desimaaleina 20 ja B on heksadesimaali 46 eli desimaaleina 70. Mainitulla yhtälöllä saadaan tulos 1297,5 rpm eli moottorin kierrosluku. Kuvassa 21 nähdään kuinka, tulos käännetään koodissa. Samankaltaisesti obd2 pid -komennon ’010D’-käännös desimaaliluvuksi on A, jossa A on 5C heksadesimaali eli desimaaleina 92.

```
func getEngineRPM()-> Int{

    /*  Response conversion to decimal, the response should be a string
        as in 7E8 04 41 01 14 46, 7E8 is the header, 04 is the byte size and
        41 01 14 46 is the response, with 14 & 46 the values we need for the
        conversion to a decimal value.
     */
    var responseData: String = "7E804411446"

    let start = responseData.index(responseData.startIndex, offsetBy: 7)
    let end = responseData.index(responseData.endIndex, offsetBy: -2)
    let range = start..<end

    let hexByte1 = responseData[range]

    let start2 = responseData.index(responseData.startIndex, offsetBy: 9)
    let end2 = responseData.index(responseData.endIndex, offsetBy: 0)
    let range2 = start2..<end2

    let hexByte2 = responseData[range2]


    var decByte1 = Int(hexByte1, radix: 16)!
    var decByte2 = Int(hexByte2, radix: 16)!

    print("hexByte1 = \(hexByte1) decByte1 = \(decByte1)")
    print("hexByte2 = \(hexByte2) decByte2 = \(decByte2)")

    /*  Formula for getting the engine rpm ((256*A)+B))/4
        where A = decByte1 and B = decByte2
     */
    engineRPM = ((256*decByte1 + decByte2) / 4)
```

Kuva 21: OBD2 PID '010C' -käännösfunktio

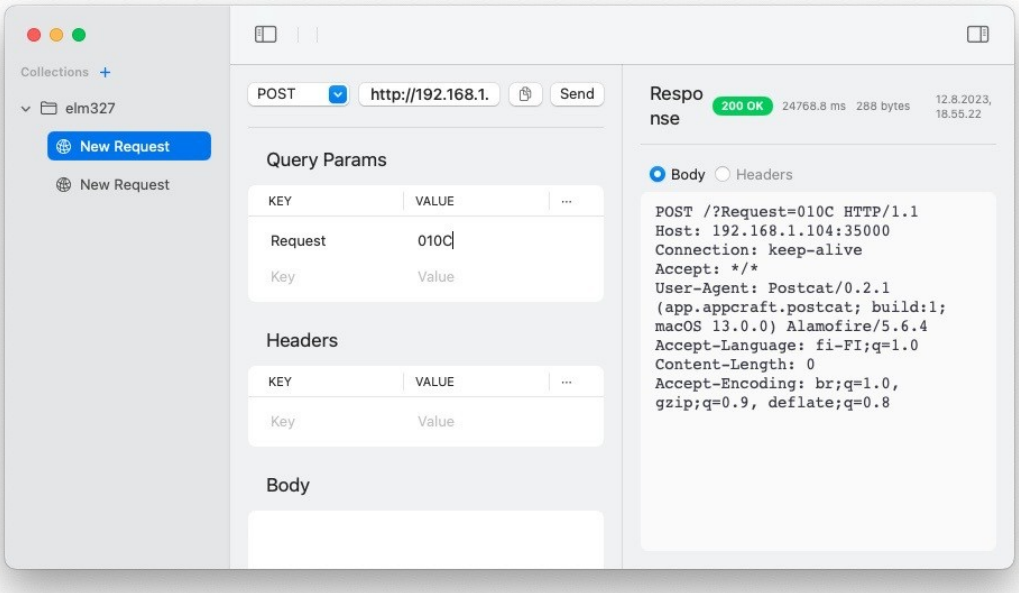
Lopullinen tulos applikaatiossa ja sen näkymä käyttöliittymässä nähdään kuvasta 22.

Kuva 22: Lopullinen sovelluksen käyttöliittymä yhdistyneessä tilassa

Tilanne oli sen mukainen, että ei ollut lainkaan dokumentaatiota tcp/ip-protokollan hyödyntämisestä tällä kolmannen osapuolen kehittämällä Python-ohjelmistolla, joten koitin aivan kokeilemalla erilaisia http-pyyntöjä PostCat-sovelluksella, jolla voidaan lähettää muun muassa POST- ja GET http-pyyntöjä sisäverkossa pyörivälle Python-sovelluksille.

Tämä menetelmä ei kuitenkaan tuottanut haluttuja tuloksia, sillä mahdollisia pyyntöjä on lukuisia määriä, joten ainut tapa olisi tutkia sovelluksen lähdekoodia, jotta selviäisi se minkälaisia http-pyyntöjä tämä sovellus odottaa lähettäjältä. Lähdekoodia on kuitenkin tuhansia rivejä koodia, joten tarvittavien resurssien määrä siihen ei ole projektin kannalta mahdollista.

Mainittakoon se kuinka, kyseenalaistava kehitysideologia Python obd2 -emulaattoriprojektilla on, sillä itse komentorivillä ei pysty syöttämään emulaattorille elm327 AT -komentoja. Sovellus kuitenkin näyttää komentorivitulosteella sen kuinka, se muuntaa obd2 pid -komennot asianmukaisiksi AT-komennoiksi.



Kuva 23: PostCat http -pyyntöesimerkki

Toisin sanoen, tämä python elm327 obd2 -emulaattori on sikäli toimiva haluamaani käyttötarkoitukseen, mutta dokumentaation puute ja kehitysalustani sopimattomuus tarkoittaa sitä, että en onnistu yhdistämään tätä emulaattoria sovellukseeni halutulla tavalla. Voisin mahdollisesti jälkikäteen ottaa tähän kehittäjään yhteyttä ja pyytää häntä täydentämään dokumentaatiota tcp/ip-protokollan käyttämisestä omissa projekteissa ja refaktoroida koodiani sen mukaan.

6 Tulokset ja jatkokehitys

Sovelluksen ja projektin lopputulos jäi ehkä hivenen alle sen mitä, olin projektin alussa ajatellut, kun projekti osoittautuikin vaativammaksi kuin mitä osasin olettaa tälläiselta projektilta. Täytyy kuitenkin sanoa, että olen tyytyväinen lopputulokseen ottaen huomioon kuitenkin sen, että tämä oli minun ensimmäinen iOS-ohjelmistokehityksen ja bluetooth-ohjelmoinnin sovellus. Varsin opettavainen projekti tämä on ollut. Tästä on erittäin hyvä jatkaa iOS-kehityksen taitojen kehittämistä.

Toinen pienen muutoksen aiheuttaneen seikan oli ohjelmistokehityksen loppupuolella. Sovelluksen testiympäristö muuttui oikeasta fyysisestä henkilö-autosta simuloituun obd2-järjestelmään. Eli lopullista valmistunutta sovellusta ei päästy kokeilemaan aiemmin testivaiheessa käytetyllä elm327 obd2 -adapterilla kytkettyyn henkilöautoon, vaan ainoastaan simuloidulla obd2-järjestelmällä. Data ja toiminnot tässä simuloidussa järjestelmässä kuitenkin vastaavat aitoa obd2-järjestelmää, joten sovellus tulisi toimia täysin samanlailla elm327 obd2 -adapterilla yhdistettynä. Jatkokehityksen kannalta ajatellen tämä ei suinkaan vie pois tältä projektilta, sillä tämä vain lisää toisen tavan yhdistää elm327 obd2 -adapteriin, aiempi bluetooth-yhteyden ylitse kulkeva data voidaan jättää hautumaan tämän projektin koodiin.

Mielestäni sovellus onnistui varsin hyvin. Saavutin ne toiminnallisuudet jotka olin projektin alussa kaavaillut, mutta kuitenkaan en juuri sillä tavalla mitä, olisin halunnut. Toki joitain toiminnallisuuksia tuli jättää pois sillä, aika-resurssit tulivat vastaan. Esimerkiksi olisin halunnut graafisen kierroslukumittarin sovelluksen käyttöliittymään. Tässä jäi ihan hyvin jatkokehitykselle tilaa. Toiminnallisuuksien lisäksi itse graafinen käyttöliittymä jäi ehkä hieman yksinkertaiseksi eli mikäli aion jatkokehittää tätä projektia, siinä on yksi työnsarka sovelluksen jatkojalostamiseksi.

Itse ohjelmointi ja ’koodaus’ tällä xcode 14-ympäristöllä ja Swift-kielellä oli varsin luontevaa ja eikä oikeastaan ollut hankalaa. Ohjelmointi oli hyvin

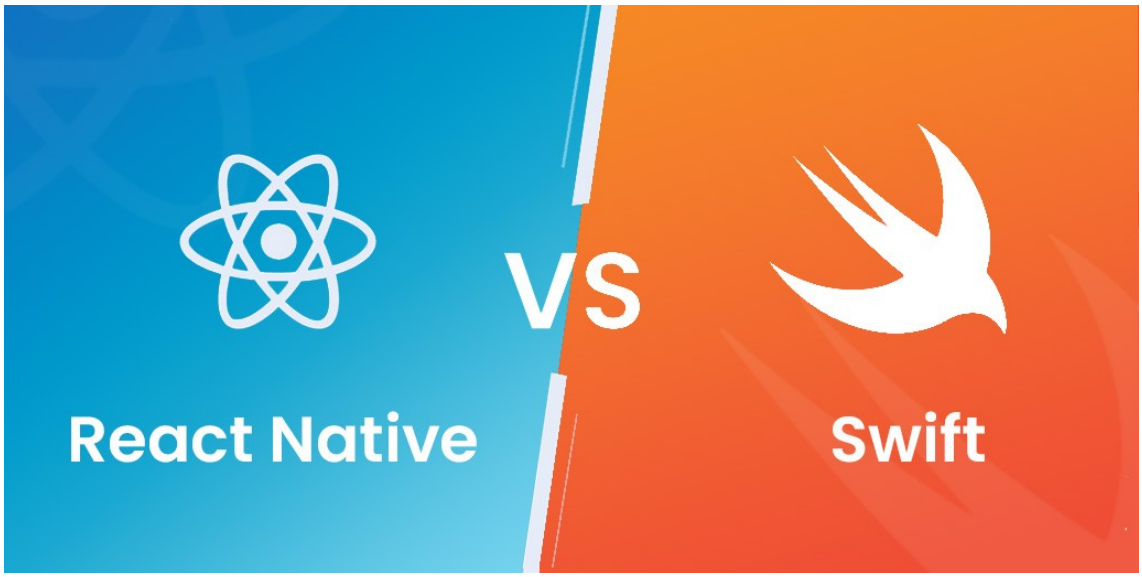
samankaltaista kuin Javalla, josta minulla on kaikista eniten ohjelmointikokemusta, eli hyvin tyypillistä olio-ohjelmointia. Appleltä löytyvät myös erittäin kattavat dokumentaatiot. Ainut heikko asia sanottava tästä kielestä on kuinka, se on vielä suhteellisen uusi ohjelmointikieli, joten apua tiettyihin ohjelmointipulmiin kolmansilta osapuolilta ei välttämättä löydy aivan niin helposti. Kuitenkin sanottakoon, että uuden oppijan/käyttäjän näkökulmastä on Swift-ohjelmointikieli todella helppoa oppia käyttämään. Koodin ajologiikka ja xcode-ohjelmointiympäristö varsinkin on erittäin auttava ja, se tuo heti esille kaikki mahdolliset ohjelmointivirheet ennen koodin kääntämistä.

Kuva 24: iOS-kehittäjä

Tämä projekti on ollut erittäin mielenkiintoinen ja, olenkin päässyt oppimaan todella paljon uutta, ja aion ehdottomasti jatkaa iOS-kehitystä ja parantaa taitojani iOS-kehittäjänä. Se on ollut jo hyvin pitkään minua kiinnostava kehitysympäristö ja, kenties voisinkin tulevaisuudessa työllistyä iOS-kehittäjänä joko toisen yrityksen leivissä tai sitten yksityisenä yrittäjänä.

Aiheesta hieman sivuten täytyy mainita, että vaikka Applen markkina-osuus ei ole lähelläkään Googlen Android-osuuksia, on Applen sovelluskehitys ja käyttöympäristöt sekä muu sovelluslogiikka aivan toisella tasolla.

Minulla on hieman aikaisempaa kokemusta Android-sovelluskehityksestä AndroidStudiolla, ja kehitysympäristönä ihan toimiva, mutta ainakin käytettävyyden kannalta Apple:n Xcode on paljon parempi. Varmasti osalta on mielipidekysymys, mutta olen vahvasti sitä mieltä, että taidot iOS-kehittäjänä ovat arvostetuimpia kuin Android-kehittäjän. Tähän voisi sivusta mainita kuinka, jotkin kehittäjät suosivat monen alustan ohjelmistokehyksiä kuten muun muassa react native, joka kylläkin täydentää ja soveltuu joihinkin kaupallisiin projekteihin, mutta kuitenkin työmarkkinoilla on tarvetta valikoiduille yhden alustan kehittäjille kuten ios-alustan tai sitten android-alustan.



Kuva 23: React native verrattuna Swiftiin

Lähteet

1 Mark du Preez and others, Johan Zandin. 2022. https://pinoutguide.com/CarElectronics/car_obd2_pinout.shtml. Verkko artikkeli. Luettu 30.3.2023.

2 ELM327 Devices. 2023. https://obd2-elm327.com/elm327-devices-information. Verkko artikkeli. Luettu 30.3.2023.

3 Lammert Bies. 2021. https://www.lammertbies.nl/comm/info/hayes-at-commands. Verkko artikkeli. Luettu 30.3.2023.

4 How Does Bluetooth Work. 2023. https://www.electronics-notes.com/articles/connectivity/bluetooth/how-bluetooth-works.php. Verkko artikkeli. Luettu 30.3.2023.

5 Jason Marcel. 2020. https://www.bluetooth.com/blog/how-bluetooth-technology-uses-adaptive-frequency-hopping-to-overcome-packet-interference/. Verkko artikkeli. Luettu 30.3.2023.

6 Swift. 2023. https://developer.apple.com/swift/. Verkko artikkeli. Luettu 30.3.2023.

7 Esat Kemel Ekren. What is Xcode and How to Use It?. 2022. https://www.netguru.com/blog/what-is-xcode-and-how-to-use-it. Luettu 30.3.2023.

8 Ircama. ELM327-emulator. 2021. https://github.com/Ircama/ELM327-emulator. Github versionhallinta säilytyspaikka. Luettu 16.6.2023.