OBD를 이용한 차량 진단 애플리케이션
<차다>

김재환 김지훈 오승은

목차

Ⅰ. 작품 개요

1-1. 개요

작품 명은 <OBD 를 이용한 차량 진단 어플리케이션 '차다'> 이다. OBD 는 ECU 의 정보를 쉽게 확인할 수 있도록 도와주는 자동차 진단 시스템이며, '차다'는 '차를 보다'의 약어로 차에 대한 정보를 볼 수 있다는 의미를 표현했다.

우리 팀이 해결하려는 것은 차량 점검 시 부당한 수리비가 청구되는 것이다. 정확한 문제 파악을 위해 정비소 이용과 관련된 공공 자료와 기사를 참고했다. 이 과정에서 부당한 수리비 외에 개인 차량 정비, 전방 주시에 대한 문제를 발견했다. 총 3 가지 문제에 대한 개선 방안을 고민하며 3 가지 기능을 개발했다.

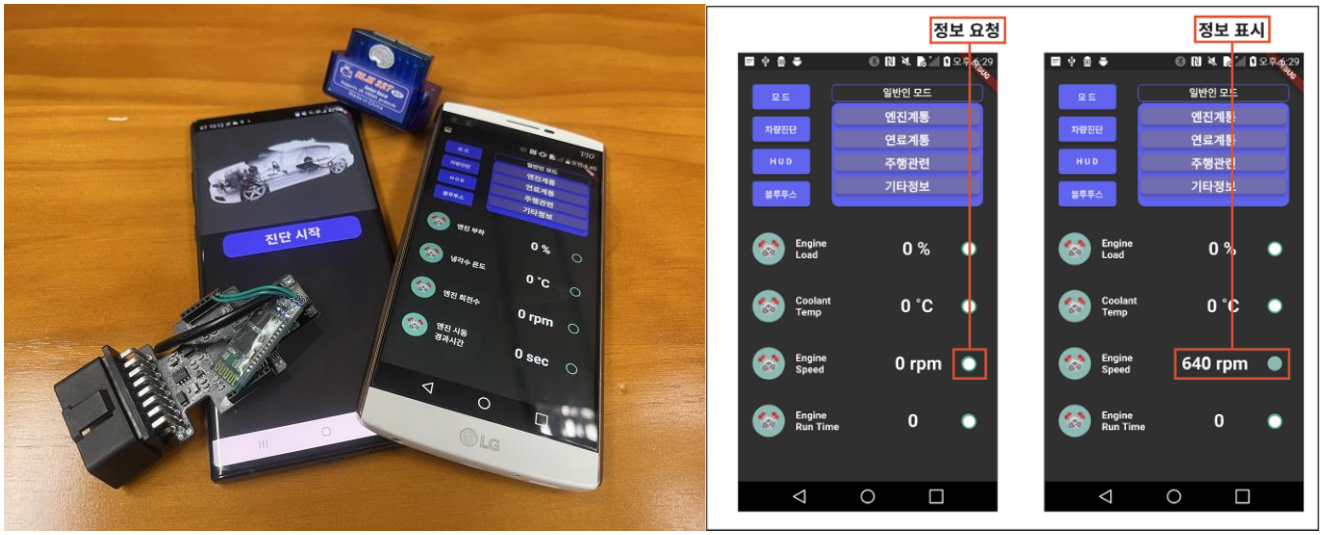
1) 차량 진단
2) 차량 정보 제공 (일반인/전문가 모드)
3) 속도 HUD

OBD(On-board diagnostics)란?

OBD는 차의 상태를 진단하고 결과를 알려주는 장치이다. 최근에 생산되는 자동차는 여러 가지 계측과 제어를 위한 센서를 탑재하고 있고, 센서들은 ECU(Electronic Control Unit)가 제어하고 있다.

ECU(Electronic Control Unit)란?

자동차의 전자 제어 장치로서 자동차의 엔진, 변속기(자동), 조향 장치, 제동 장치 등에 대한 정보를 관리 및 제어하는 장치이다.



작품 사진

1-2. 작품 필요성

1-2-1. 차량 진단

■ '과도한 수리비 청구'(47.8%), '차주동의 없이 임의수리'(22.2%) 피해 많아

부당 수리비 180건에 대한 피해유형을 세부적으로 살펴보면
'과도한 수리비 청구'가 86건(47.8%)으로 많이 차지함. 다음으로 '차주동의 없는 임의수리' 40건(22.2%), '과잉정비' 29건(16.1%), '수리하지 않은 비용 청구'가 25건(13.9%)임

정비업자는 수리하기 전 소비자에게 수리범위, 수리비용 등에 대한 '자동차점검·정비견적서'를 발급하지 않거나, 수리과정 중 추가정비가 필요한 경우 소비자와 수리범위 및 수리비에 대해 추가 협의가 이루어지지 않다보니 관련 분쟁이 발생하고 있음.

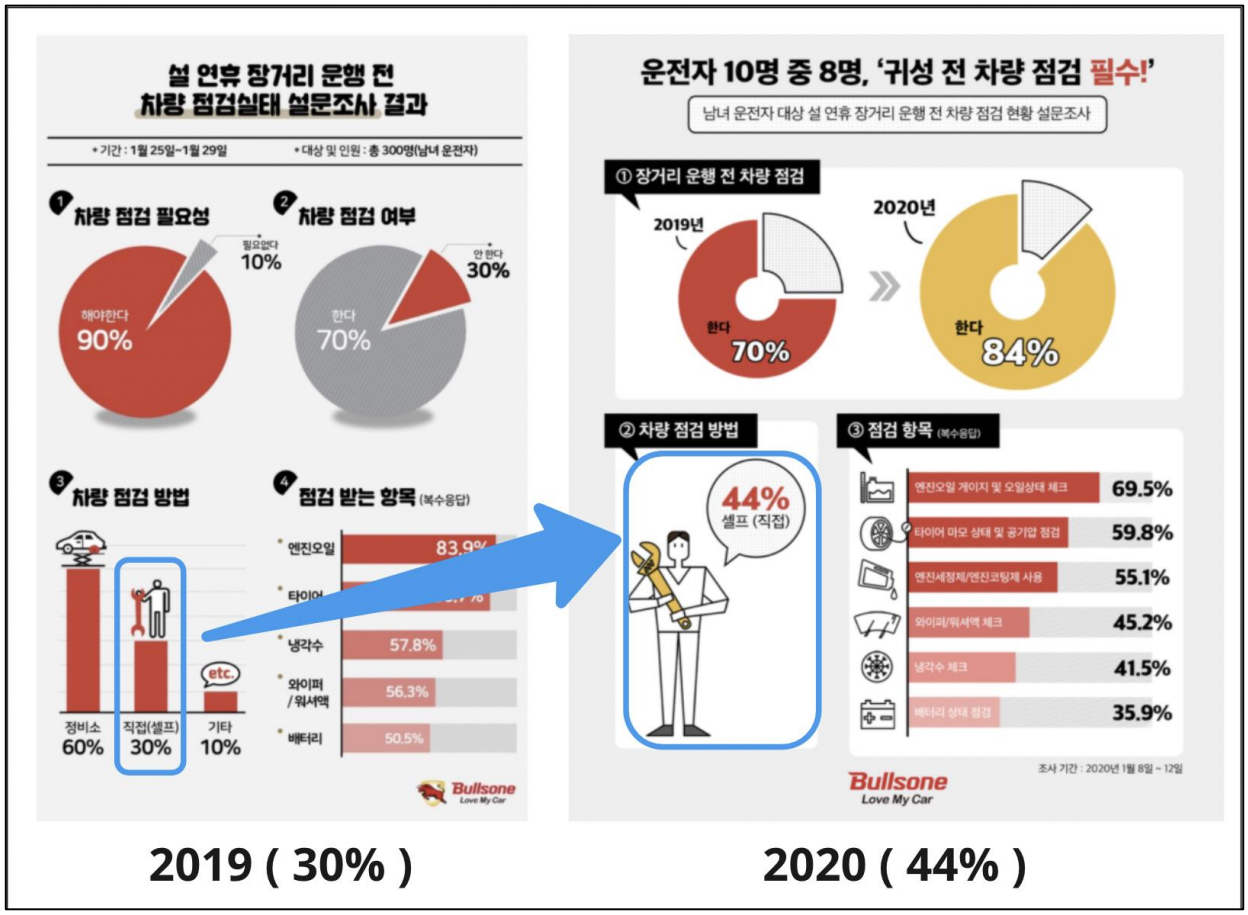
< 부당수리비 피해 유형 >(단위: 건, %)

구 분	과도한 수리비 청구	차주동의 없는 임의수리	과잉정비	수리하지 않은 비용 청구	계
건수(%)	86(47.8)	40(22.2)	29(16.1)	25(13.9)	180(100.0)

부당수리비에 대한 자료 <출처 한국소비자원>

차량 진단 기능은 부당 수리비 청구로 인한 소비자 불만과 피해를 예방할 수 있다. 한국소비자원 소비자뉴스에 따르면 부당 수리비, 부족한 수리 등 자동차정비 관련 소비자 피해가 끊임없이 발생하고 있으며 소비자들의 주의가 필요하다고 한다. 부당 수리비 유형을 보면 과도한 수리비 청구와 과잉 정비가 47.8%, 29%로 높은 것을 확인할 수 있다.
이 문제는 운전자가 자동차의 어떤 부분에 문제가 생겼는지 모르기 때문인 것을 확인했다. 차량 진단은 현재 자동차의 어떤 부분이 고장났는지 파악하는 기능이다. 운전자는 차량 진단 기능을 사용함으로써 자동차의 상태를 미리 인지하고, 정비사의 부당 수리비 청구를 예방할 수 있다.

1-2-2. 차량 정보 제공



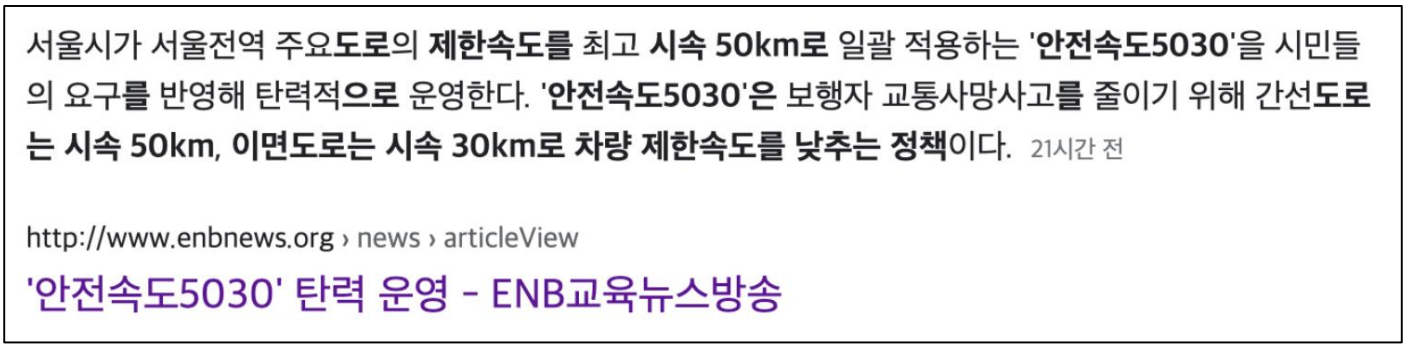
2019 년, 2020 년 차량 점검 실태 조사 (직접 차량 점검하는 운전자 14% 증가) <출처 불스원>

내 차 구매에 대한 수요 증가 기사 <출처 이코노믹뷰>

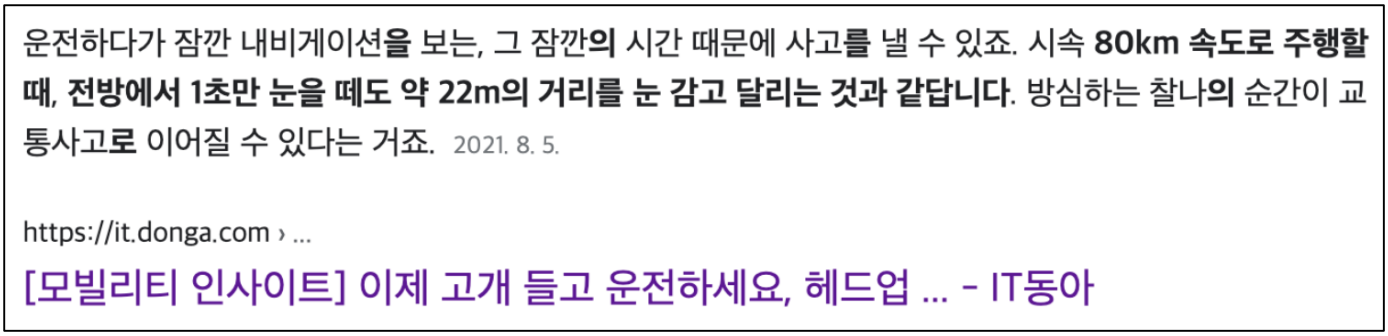
차량 정보 제공 기능은 효과적인 차량 점검을 가능하게 할 것이다. 불스원 차량 점검 실태 설문조사에 따르면 장거리 운행 전, 차량 점검을 하는 사람이 증가하고 있다. 직접 차량 점검을 하는 사람은 2019 년 30%에서 2020 년 44%로 상승했다. 두 번째 기사는 코로나 19 로 인해 대중교통 이용이 줄고, 내 차 마련에 대한 수요가 늘고 있다고 한다.
위 내용을 바탕으로 자가 자동차를 소유한 운전자와 셀프 차량 점검을 하는 운전자가 앞으로도 증가할 것이라고 판단했다. 차량 점검 시 엔진/엑셀/연료 등에 대한 정보가 필요한데, 차량 정보 제공 기능은 이러한 정보를 실시간으로 제공하여 효과적인 차량 점검을 할 수 있다.

1-2-3. 속도 HUD

속도 HUD 기능은 특정 장소나 상황에서 속도 확인을 위해 눈 돌리는 것을 방지함으로써 전방 주시와 안전 운행에 도움을 줄 수 있다. 아래 기사는 최근 정부가 이면 도로와 간선도로의 제한 속도를 조정했고 한다. 또한 시속 80km 에서 1초 눈을 떼는 것은 22m의 거리를 눈을 감고 달리는 것과 같다고 한다. 이는 40km 주행이라고 해도 11m의 거리를 눈 감고 가는 의미이다.
바뀐 제한 속도에 적응하는 과정에서 운전자는 계기판을 자주 확인할 것이고, 속도는 지키지만 전방 주시에 소홀할 수 있다고 판단했다. HUD 기능은 속도를 전방에 보여줌으로써 전방 주시와 안전 운행에 도움을 줄 수 있다.



변화된 속도 제한에 대한 기사 <출처 ENB교육뉴스방송>



전방 주시에 대한 중요성에 관한 기사 <출처 IT 동아>

1-3. 향후 목표

1) 점검 및 소모품 교체 알림 서비스

차다는 자동차가 현재 필요한 점검이나 교체해야 할 소모품을 알려주는 기능을 고려하고 있다. OBD를 통해 얻을 수 있는 정보를 활용하면, 주요한 정기 점검이나, 소모품에 대한 교체 주기 알림을 자동화 할 수 있다.

많은 차량이 서비스를 이용한다면, 각종 소모품이나 정비에 대한 수요의 흐름을 파악할 수 있다. 이런 정보를 통해 정비업체가 정비나 소모품에 대한 견적을 오픈마켓처럼 제시할 수 있고, 차주가 원하는 업체를 선택할 수 있는 플랫폼 서비스까지 확장하는 것을 기대하고 있다.

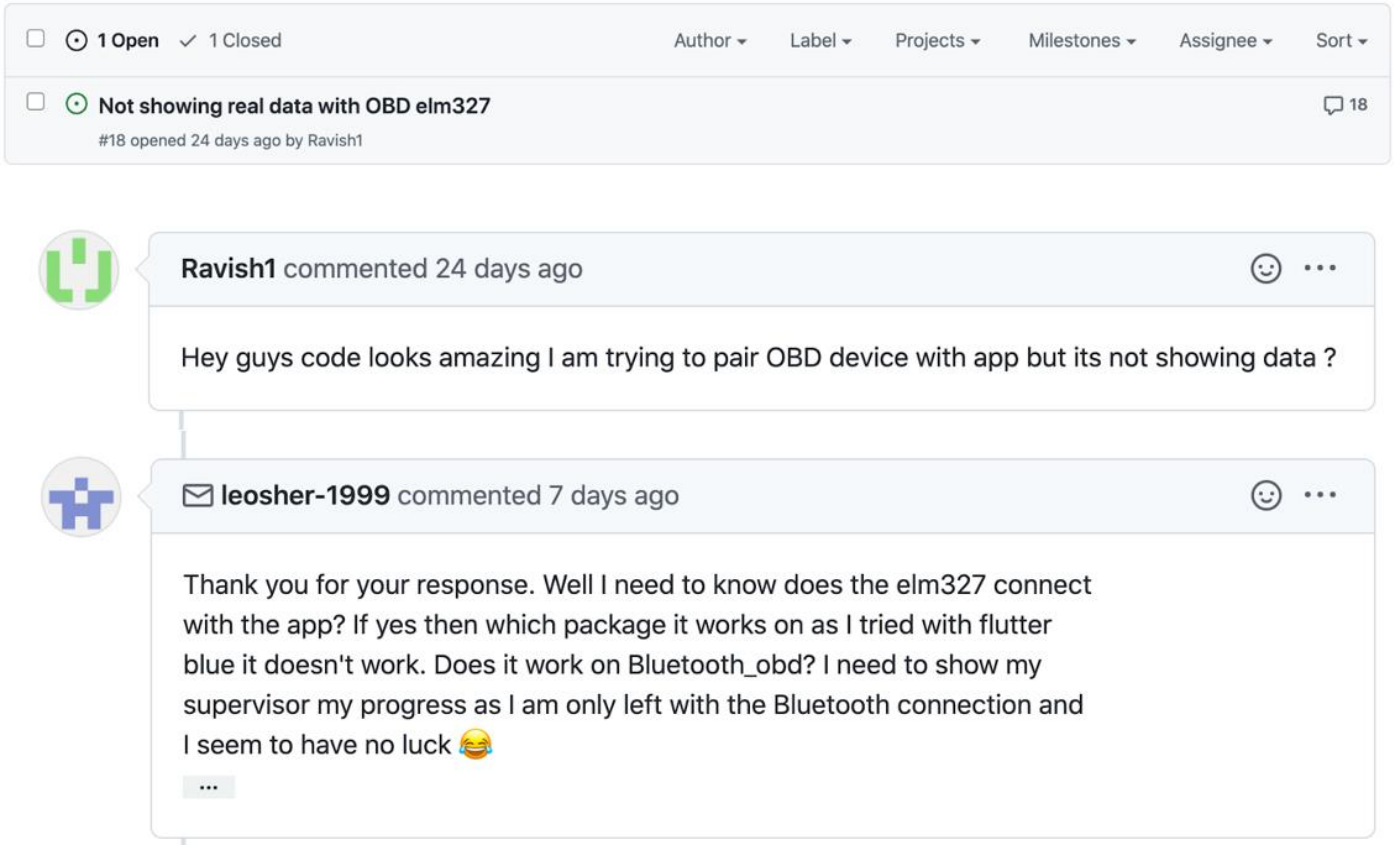
1) 블록체인 기반 중개인 없는 중고차 거래 플랫폼

차다는 블록체인 기술을 고려하고 있다. 이는 중고차 거래 시 유용하게 이용될 것이라고 기대한다. 현재 중고차 성능 상태 등이 실제 거래 내용과 달라 피해를 보는 사례가 끊임없이 발생한다. 이러한 문제에 블록체인 기술을 도입한다면 정보의 무결성을 확보할 수 있을 뿐만 아니라 중개인 없이 판매자와 구매자가 직접 거래를 할 수 있기 때문에 피해를 예방할 수 있다.

1-4. 기타 (개발 진행 과정에서 '차다' 서비스에 대한 수요 발생)

프로젝트를 진행하는 과정에서 프로젝트에 대한 수요를 확인할 수 있었다. 당시 진행하던 모듈(OBD-II CAN BUS GPS Develpoment Kit)이 아닌, 다른 모듈(ELM327)과 호환이 되는 애플리케이션을 원하는 것이었다.

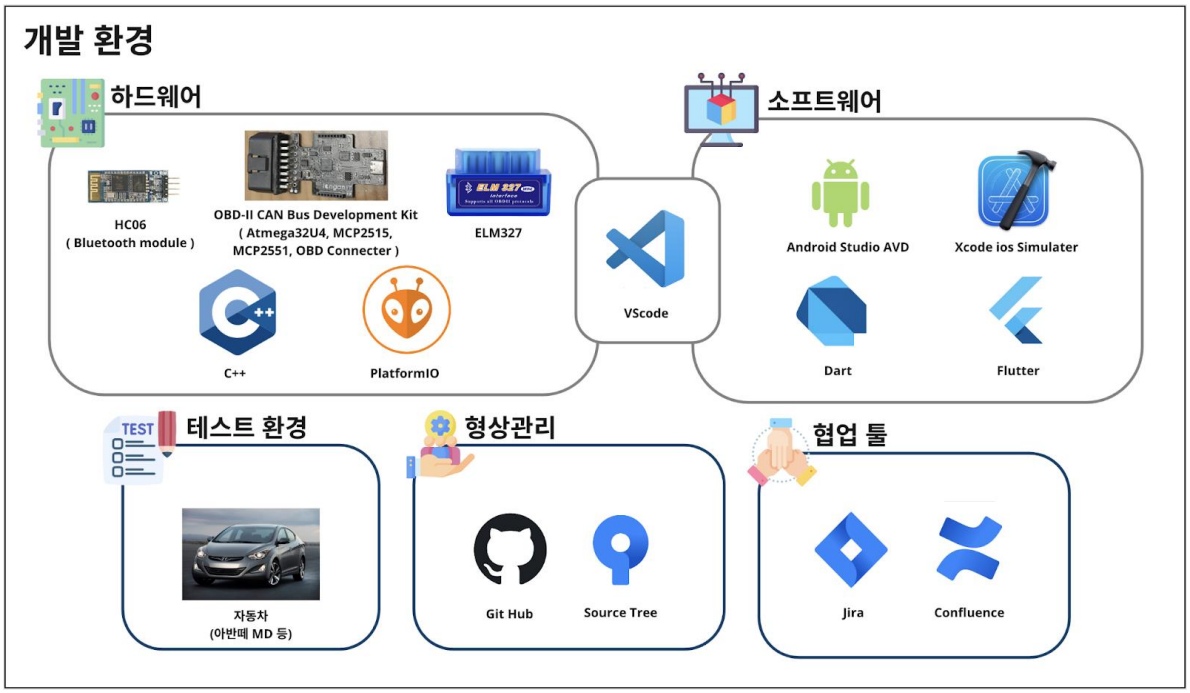
우리는 수요를 확인하며 애플리케이션의 호환성과 확장성을 고려했고, 진행 중인 모듈뿐만 아니라 ELM327을 통한 차량 진단 장치 개발도 함께 진행했다.



GitHub이슈(https://github.com/komskoms/OhJiHwan_CAN_OBD)

II. 작품 설명

2-1. 개발 환경

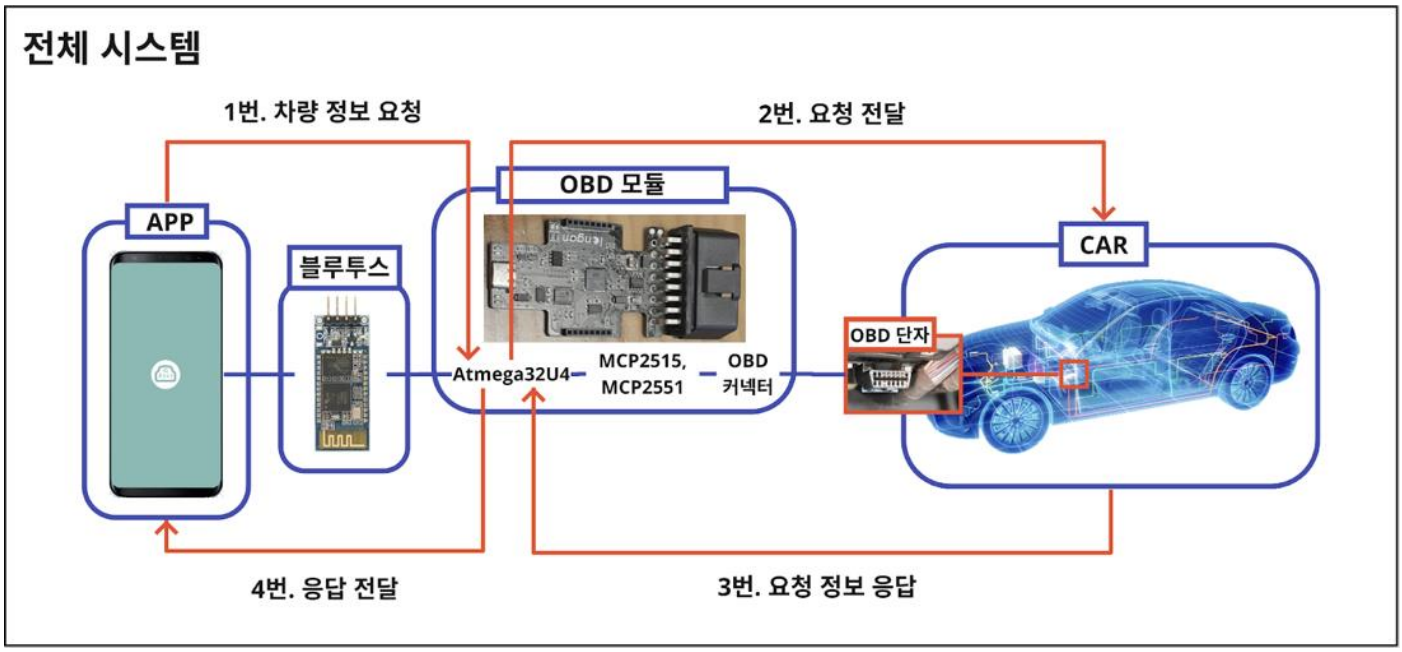


개발 환경

하드웨어 (OBD 차량 진단 장비)		소프트웨어 (애플리케이션)	
MCU	ATmega32U4	**프레임워크**	Flutter (2.10.2)
언어	CPP (C++11)	**언어**	Dart (2.16.1)
개발 환경	VSCode (1.65.0)	**개발 환경**	VSCode (1.65.0)
	PlatformIO (3.4.1)		Android Studio AVD 개발 버전 : Gradle 6.7 에뮬레이터 : API 31, API 24, API 23
사용 모듈	OBD-II CAN Bus Development Kit ● OBD Connecter (차량 진단 장치 연결부) ● MCP2551 (CAN 통신 모듈) ● MCP2515 (CAN 통신 모듈) ● HC06 (블루투스 통신 모듈)		Xcode IOS Simulator 개발 버전 : CocoaPods 1.11.2 에뮬레이터 : IOS 15.2
		E T C	
		테스트 환경	자동차
		형상관리	Git Hub, Source Tree
	ELM327 (OBD 모듈)	**협업툴**	Jira, Confluence

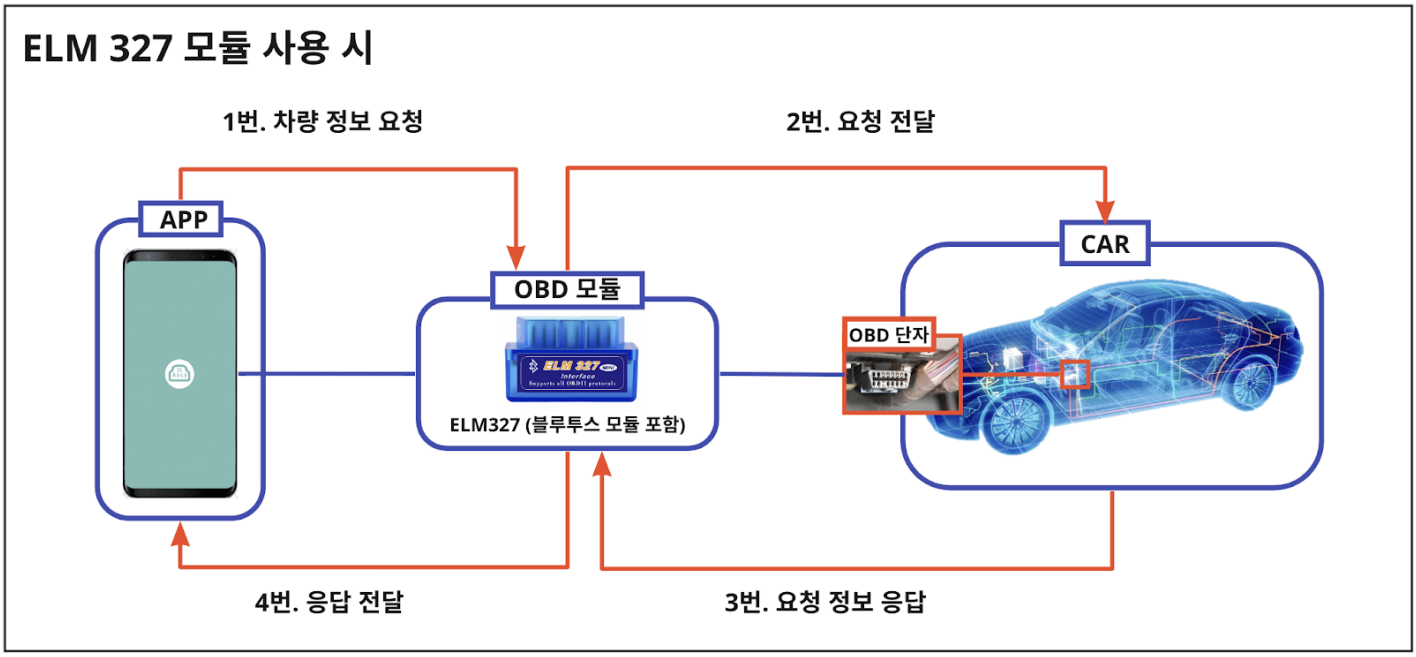
2-2. 전체 시스템 구성

차량 진단 앱 "차다"는 OBD 모듈(차량 진단기)을 자동차와 연결하여 차량 정보를 주고 받는다. 통신은 블루투스, SPI, CAN 통신을 사용한다. 정보 요청 및 응답을 통해 차량 진단, HUD, 계통별 정보 확인 기능을 사용자에게 제공한다.



전체 시스템(1)

ELM 사용 시 다음과 같다.



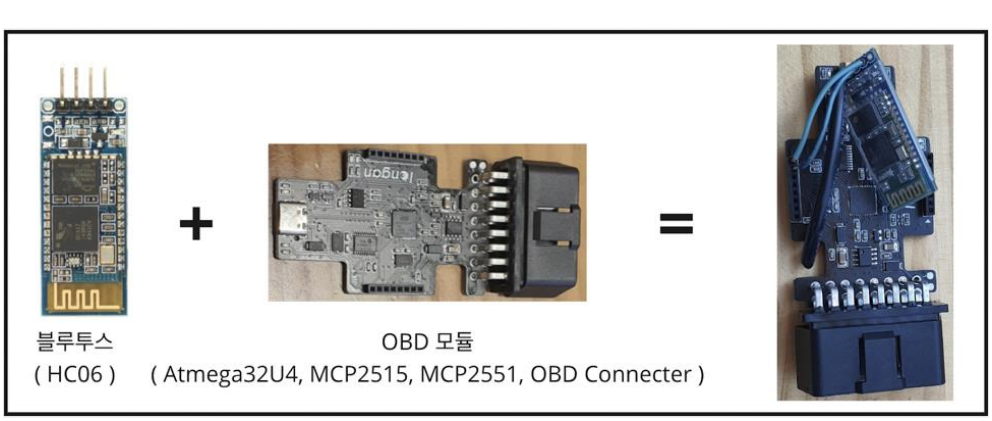
전체 시스템(2)

전체 시스템		
	이름	**설명**
통신	블루투스	스마트폰 앱과 Atmega32U4 통신
	SPI	Atmega32U4 과 MCP2515, MCP2551 통신
	CAN	MCP2515, MCP2551 과 자동차 ECU 통신
제품	자동차 (아반떼 MD)	ECU 가 차량 상태에 대한 정보를 갖고 있음
	OBD-II CAN Bus GPS Development Kit	자동차 데이터를 추출하는 차량 진단 모듈 (Atmega32U4, MCP2515, MCP2551, OBD Connector)
	HC06 (블루투스)	추출된 데이터를 스마트폰 앱에 전달하는 모듈
	ELM327	블루투스와 OBD 기능이 함께 있는 진단 장비
주요 기능	차량 진단	DTC(차량 문제 코드)를 통해 자동차 상태를 진단
	차량 정보 제공	차량 정보를 제공 1) 일반인 모드 : 일반인이 쉽게 파악할 수 있는 정보 2) 전문가 모드 : 차량 상태에 대한 자세한 정보
	HUD (Head up Display)	핸드폰 화면을 자동차 앞 유리창에 반사하여 실시간 속도 정보 제공

2-2-1. 하드웨어

2-2-1-1. OBD 모듈(차량 진단 장치)

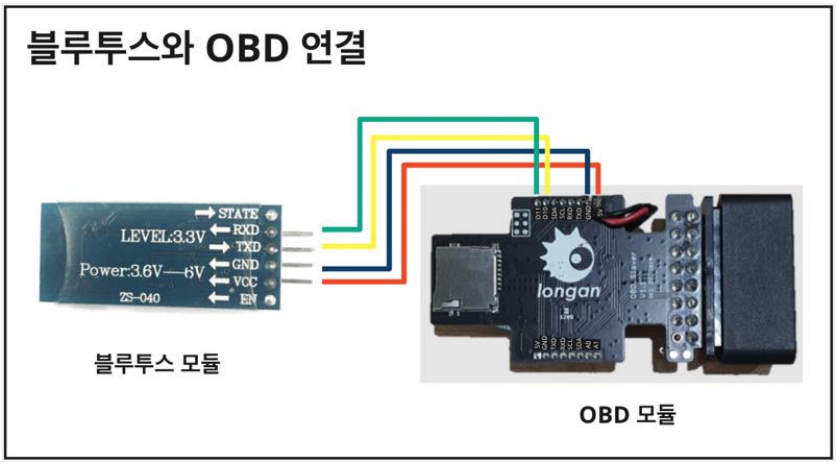
OBD 모듈에 블루투스 모듈을 결합했다. 모듈을 자동차 OBD 단자에 꽂으면 자동차 정보를 읽어오며, 읽어온 정보는 블루투스 통신을 통해 핸드폰에 전달되도록 한다. OBD 단자는 16 핀 시리얼 통신 규격과 호환되며, 핀맵은 아래와 같다. 6 번과 14 번 핀을 통해 자동차의 CAN 통신 프로토콜에 접근할 수 있으며, 이를 통해 차량 정보를 주고 받는다.



OBD 모듈, OBD-II Connector and Pinout

2-2-1-2. 핀 및 보드

핀연결	
블루투스	**OBD**
RXD	D11(TXD)
TXD	D10(RXD)
GND	GND
VCC	5V



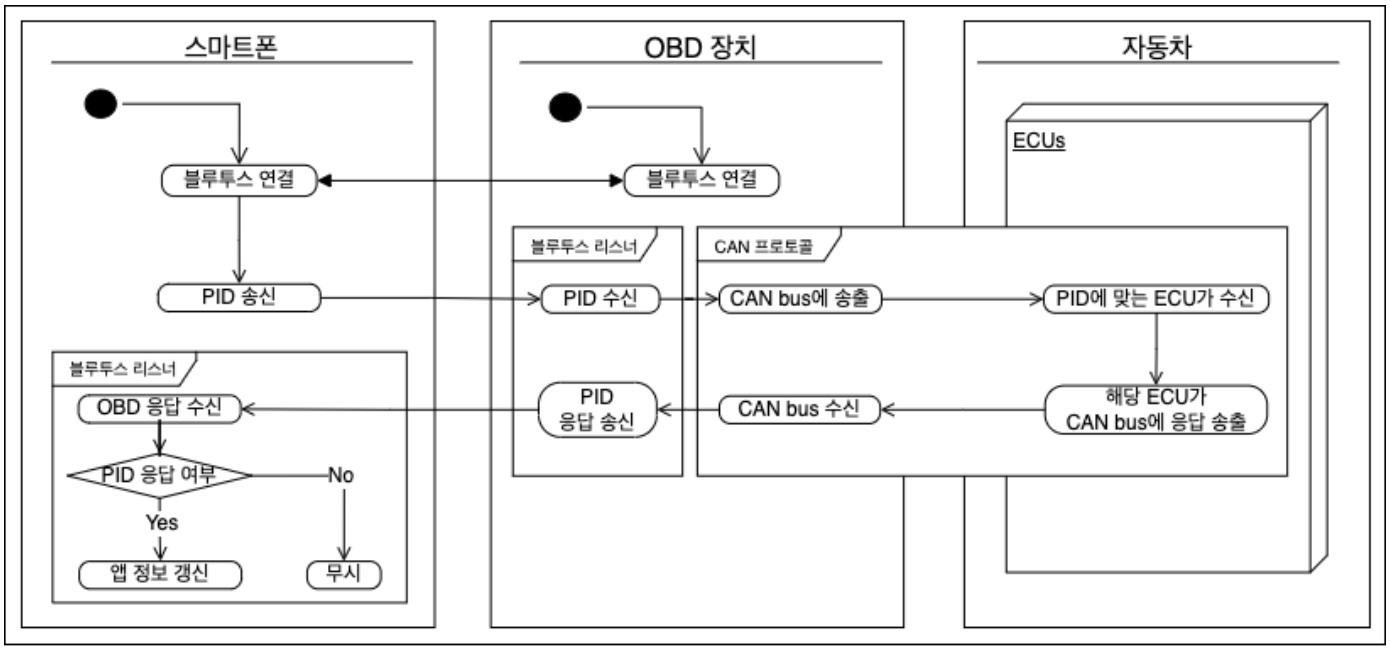
핀 보드

2-2-1-3. ELM327 (차량 진단 장치)

ELM327 은 완제품으로 하드웨어부분에 대해 별도의 작업이 필요하지 않다.

2-2-2. 소프트웨어

2-2-2-1. 애플리케이션 주요 동작 및 특징



주요 동작 및 특징

2-2-2-2. OBD 모듈 (차량 진단 장치)

- 스마트폰에서 요청한 PID 를 CAN 프로토콜을 통해 자동차에 전달
- 자동차에서 hex 값(16 진법)으로 응답하는 값을 가독성 좋게 수정
- 수정한 값을 블루투스 통신을 통해 애플리케이션에 전달

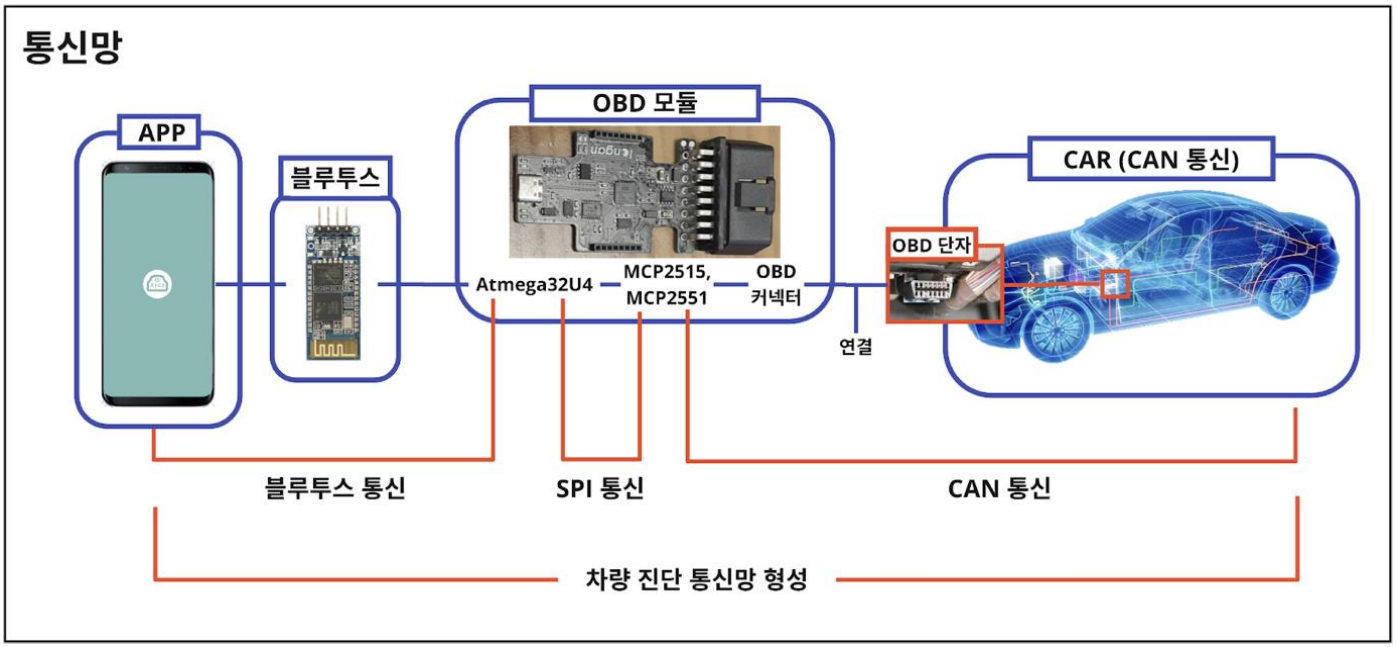
다음은 자동차로부터 응답 받은 값을 특정 수식을 통해 가독성 좋게 수정하는 코드이며, 엔진 RPM 에 대한 예시이다.

```
bool getEngineRPM(SoftwareSerial &_HC06) {
        sendPid(ENGINE_SPEED);
        unsigned long timeout_ = millis();

        while (millis() - timeout_ < 1000) {
                unsigned char len = 0;
                unsigned char buf[8];
                int rpm;
                if (CAN_MSGAVAIL == CAN.checkReceive()) {
                        CAN.readMsgBuf(&len, buf);
                        // 0x40 + 1(mode)
                        if (buf[1] == 0x41) {
                                // Serial.println(buf[3]);
                                // Serial.println(buf[4]);
                                rpm = buf[3] * 16 * 16;
                                rpm += buf[4];
                                _HC06.println("ENGINE_SPEED"+ ":" + (rpm / 4));
                                return 1;
                        }
                }
        }
        return printTimeout((char *)pidName.c_str(), _HC06);
}
```

2-2-3. 네트워크

블루투스, SPI, CAN 통신이 하나의 통신망을 형성한다. 블루투스 통신은 HC06 블루투스 모듈을 Atmega32U4 와 연결하여 형성한다. SPI 통신은 Atmega32U4 와 MCP2515, MCP2551 을 연결하여 형성한다. CAN 통신은 OBD 단자에 OBD 커넥터를 연결하며 MCP2515, MCP2551 과 통신망을 형성한다.



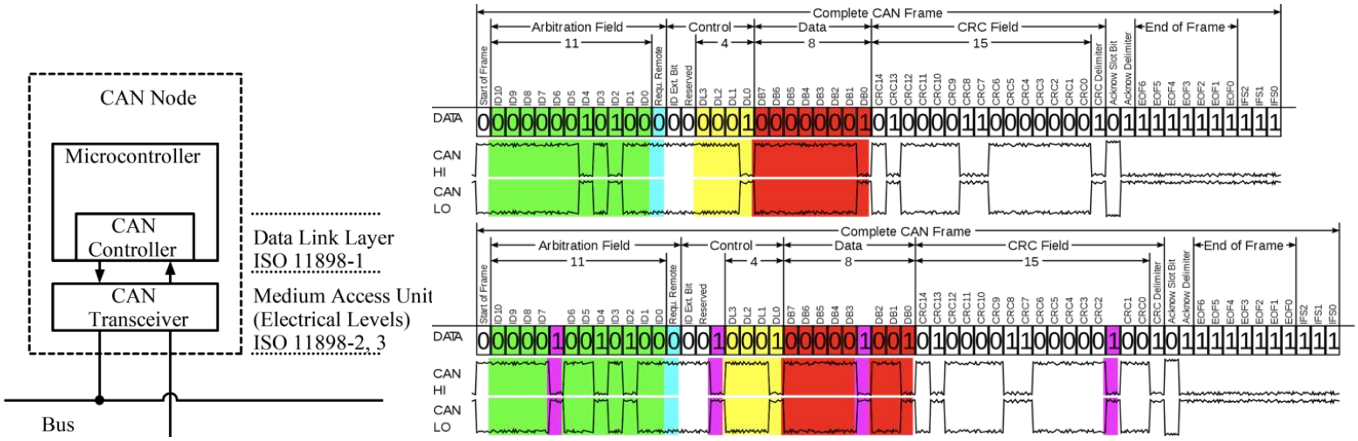
통신망

2-2-3-1. CAN 통신

CAN 은 실시간 제어를 효율적으로 지원하는 직렬 통신 프로토콜의 하나로 차량의 안전성, 편리성, 연비 등과 같은 기능을 향상시킨다. CAN 의 특징을 살펴보면, 제어 노드를 버스 형식으로 접속하고 있어서 어떤 노드에서라도 메시지 송신이 가능한 멀티마스터 방식의 버스 구성을 가지며, 송신 메시지에 식별자를 붙여 버스 상의 모든 노드에 브로드캐스팅 방식으로 메시지를 송신하여 수신측에서는 식별자를 보고 그 메시지가 자기에게 필요한 것인가를 판단한다.

CAN 은 통신을 위해 데이터 프레임, 원격 프레임, 에러 프레임, 오버로드 프레임, 인터프레임의 5 가지 형태의 프레임을 제공한다. 데이터 프레임은 데이터를 전송할 때 CAN 컨트롤러가 전송 하는 프레임이며 원격 프레임은 데이터를 요구할 경우에 사용되며 데이터 프레임에서 데이터 영역만 제외된 것이다. 에러프레임은 감지된 에러를 알리는데 사용되며 오버로드 프레임은 연속된 프레임 사이에서 지연시간을 늘리기 위해 사용된다.

본 프로젝트에서는 자동차와 OBD 모듈이 데이터를 주고 받을 때 CAN 통신을 사용한다. 왼쪽 이미지는 CAN 노드이다. 오른쪽 이미지는 CAN 컨트롤러가 전송하는 메시지 프레임이다.

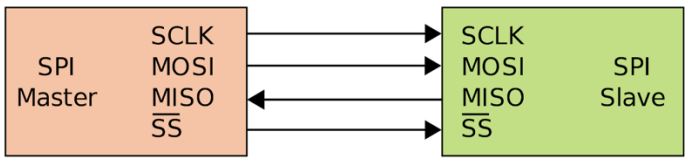


CAN NODE, CAN 컨트롤러 전송 메시지 프레임

2-2-3-2. SPI 통신

SPI 통신은 Motorola 에 의해 개발된 전이중 통신이 가능한 동기 통신이며, 다른 통신보다 단순하고 우수한 성능으로 선호도가 높은 편이다. SPI 통신의 동작 방법은 다음과 같다. 처음 SS(Slave Select)를 Low 신호로 출력하여 Slave 를 선택하여 활성화한다. 다음으로 SCLK(Serial Clock)를 제공하여 동기화하고 이 클럭에 MOSI(Master Out Slave In)나 MISO(Master In Slave Out)핀으로 데이터를 송수신하여 통신을 할 수 있는 프로토콜이다.

본 프로젝트에서는 OBD 모듈에서 Atmega32U4 와 MCP2515 및 MCP2551 이 데이터를 주고 받을 때 SPI 통신을 사용한다.



SPI 통신

2-3. 주요 기능 및 작동 방식

통신망이 구축되면, 스마트폰과 자동차는 PID 코드를 주고 받으며 차량 정보를 요청하고 응답한다. 이를 통해 차량 진단 앱 "차다"는 일 반인/전문가 모드, 차량 진단, HUD 기능을 제공한다. 기능을 사용하기에 앞서 블루투스를 연결해야한다.

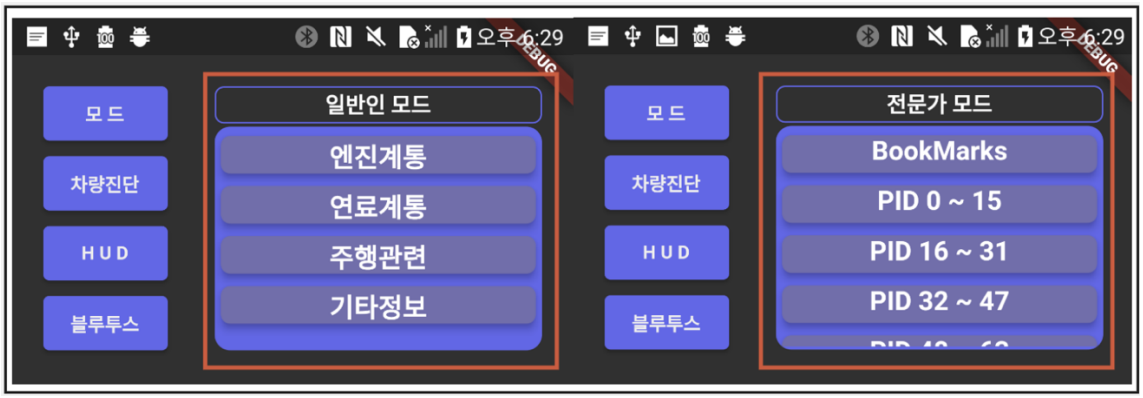
블루투스 설정 플로우 차트

> **PID란?**
> 특정 차량 데이터에 대한 코드이다. 자세한 코드는 <모드별 제공 목록> 표에 있다.

2-3-1. 일반인/전문가 모드

일반인/전문가 모드 UI 설명

모드에 따라 제공하는 정보가 달라진다. 일반인 모드에서는 일반인이 충분히 의미를 파악할 수 있을 정보들을 적절히 계통별로 그룹화하여 제공한다. 전문가 모드에서는 제공하는 PID 를 16 개 단위로 묶어 그룹화하여 표시한다. 그룹을 선택하면 화면의 하단에, 해당 그룹에 속한 항목들과 그 값이 표시된다. 또한 사용자가 자주 보는 데이터를 북마크로 설정할 수 있다.



일반인/전문가 모드 제공 정보

현재 선택된 PID 그룹의 하위항목들이 표시된다. 시인성 향상을 위한 아이콘, 항목 이름, 현재 값, 정보 요청 버튼이 순서로 배치되어 있다. 갱신 여부 설정 버튼을 비활성화하면 해당 항목은 갱신을 하지 않고, 대신 다른 항목들의 갱신 속도가 다소 향상된다. 아래 이미지는 모드별 제공하는 정보와 UI 설명 및 동작 방식이다.

일반인/전문가 모드 플로우 차트

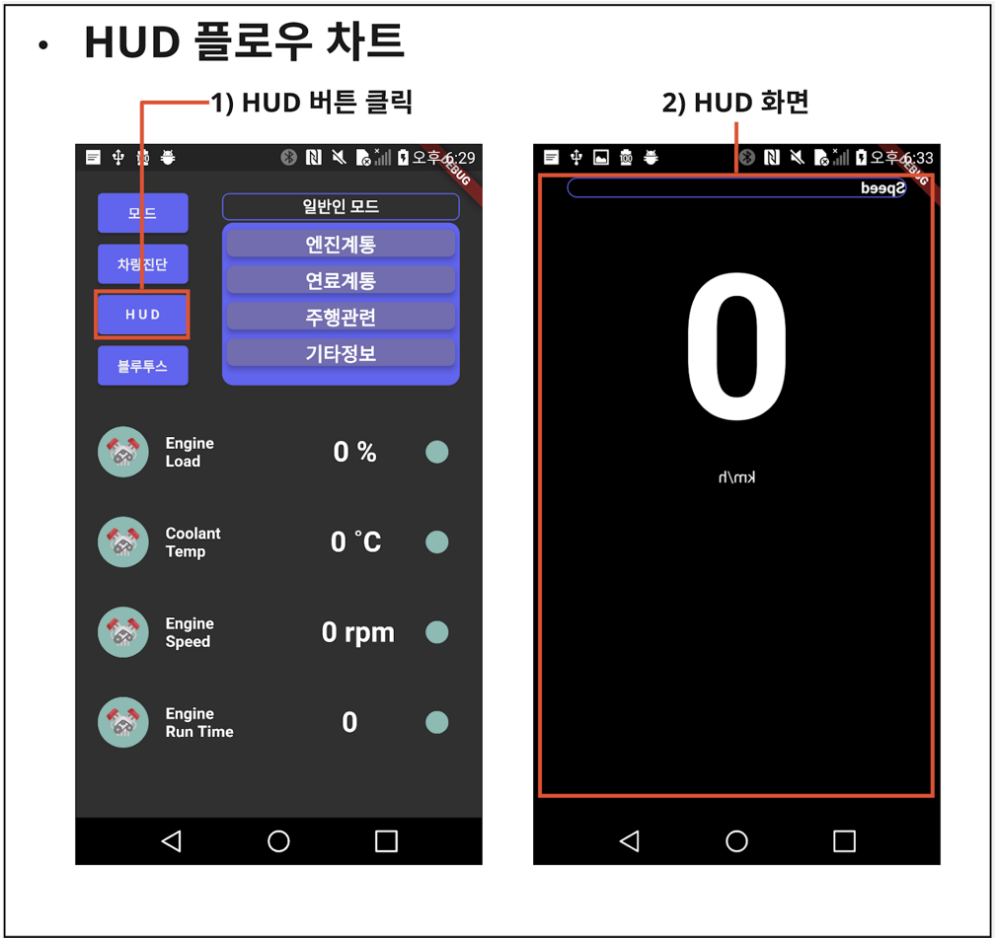
2-3-2. 차량 진단

화면 중단에 진단하기 버튼을 누르면 차량의 자기 진단 정보를 요청한다. 응답 정보를 통해 차량의 이상 여부를 확인한다.

차량 진단 플로우 차트

2-3-3. HUD (Head Up Display)

OBD 연결 후 주행을 하면, 속력이 자동 출력된다. HUD 는 자동차 전면 유리를 통해 속력을 확인할 수 있게 하는 기능이다. 유리창에 반사된 스마트폰 화면을 보기 때문에 UI 가 좌우 반전되어있다.



HUD 플로우 차트

모드별 제공 목록					
모드	**계통**	**PID (DEC)**	**제공 정보**	**단위**	**ECU 데이터 가공 수식**
일반인 모드	엔진	4	엔진 부하 Calculated engine load	%	100 / 255 * A
		5	냉각수 온도 Engine coolant temperature	°C	A - 40
		12	엔진 회전 속도 Engine speed	rpm	(256 * A + B) / 4
		31	엔진 시동 후 시간 Run time since engine start	seconds	256 * A + B
		92	엔진 오일 온도 Engine oil temperature	°C	A - 40

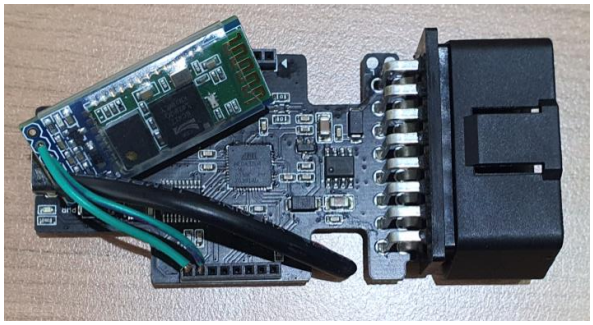
	연료	10	연료 압력 Fuel pressure	kPa	3 * A
		47	연료 탱크 충전 용량 Fuel Tank Level Input	%	100 / 255 * A
		94	엔진에 주입되는 연료량 (시간) Engine fuel rate	L/h	(256A + B) / 20
	엑셀 & 속력	13	속력 Vehicle speed	km/h	A
		17	스로틀 위치 Throttle position	%	100 / 255 * A
		90	가속 페달 위치 Relative accelerator pedal position	%	100 / 255 * A
	ETC	33	경고등 켜진 상태의 주행거리 Distance traveled with malfunction indicator lamp (MIL) on	km	256 * A + B
		66	배터리 전압 Control module voltage	V	(256 * A + B) / 1000
		78	차량 고장 코드 삭제 후 경과 시간 Time since trouble codes cleared	minutes	256 * A + B
		91	하이브리드 배터리의 남은 수명 Hybrid battery pack remaining life	%	100 / 255 * A
전문가 모드	PID 00 15		하단 상세 정보 페이지 참고		
	PID 16~31		하단 상세 정보 페이지 참고		
	PID 32~47		하단 상세 정보 페이지 참고		
	PID 48~63		하단 상세 정보 페이지 참고		
	PID 64~79		하단 상세 정보 페이지 참고		
	PID 80~95		하단 상세 정보 페이지 참고		

상세 정보 페이지 : https://en.wikipedia.org/wiki/OBD-II_PIDs

2-4. 최종 구현 및 시연

2-4-1. 구현

2-4-1-1.하드웨어 (OBD 모듈)



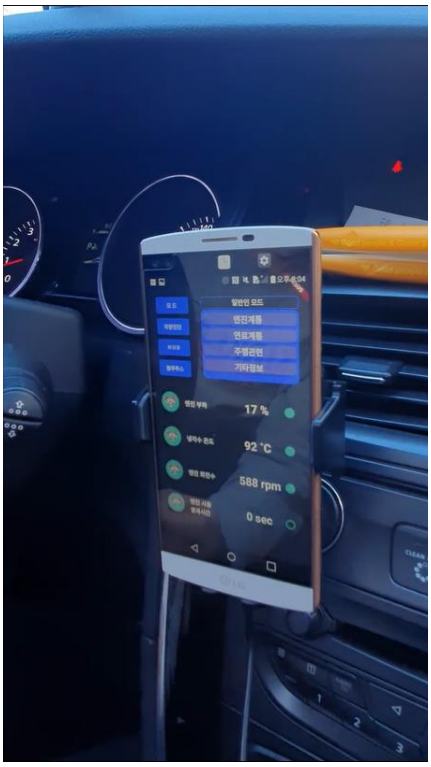
하드웨어

2-4-1-2. 소프트웨어 (UI/UX)

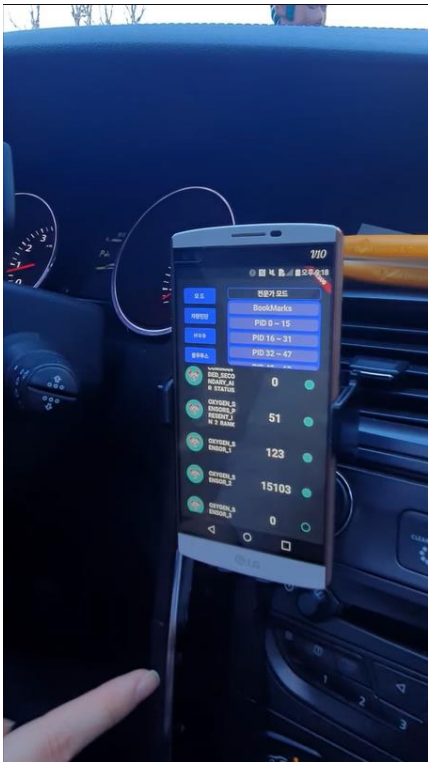
APP UI/UX

2-4-2. 시연

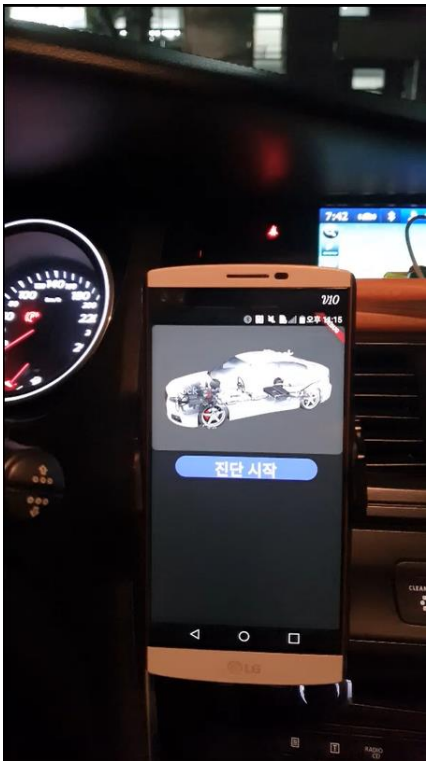
2-4-2-1. 일반인 모드



2-4-2-2.전문가 모드

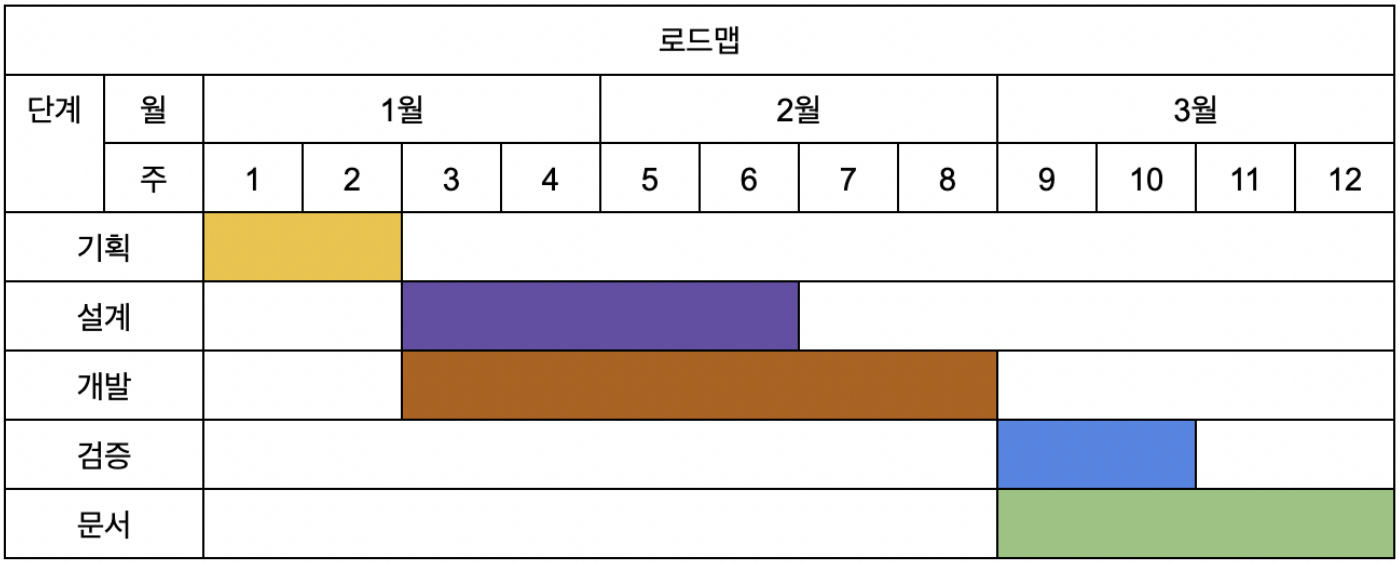


2-4-2-3. 차량 진단



2-4-2-4. HUD

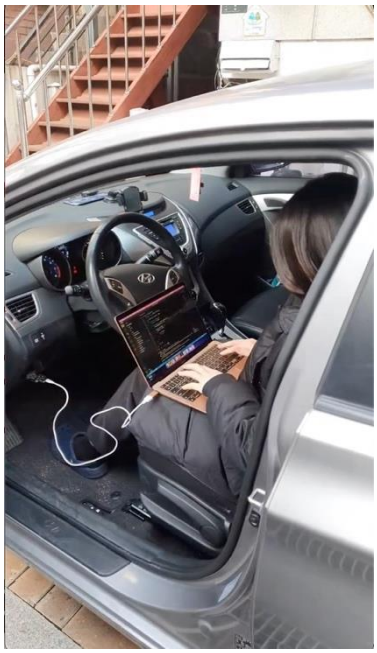
III. 단계별 제작 과정

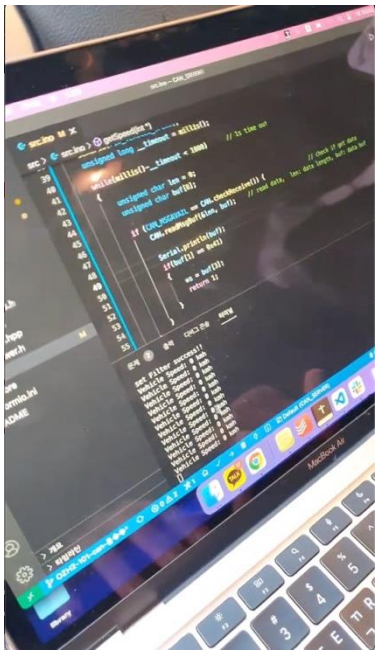


1 주	프로젝트 선정 및 기획	7 주	스마트폰 앱 UI/UX 구현 및 기능 구현
2 주	프로젝트 기획 및 요구사항 분석	8 주	스마트폰 앱 기능 및 연동 인터페이스 구현
3 주	HW, SW 연동 인터페이스 설계	9 주	연동 인터페이스 구현 및 테스트
4 주	OBD 모듈 통신부 API 설계	10 주	테스트, 수정 및 문서 작업
5 주	OBD 모듈 통신부 API 구현	11 주	테스트, 수정 및 문서 작업
6 주	플러터 학습 및 스마트폰 앱 UI/UX 설계	12 주	문서 작업

3-1. OBD 모듈 이상 유무 확인(시리얼 모니터, 자동차)

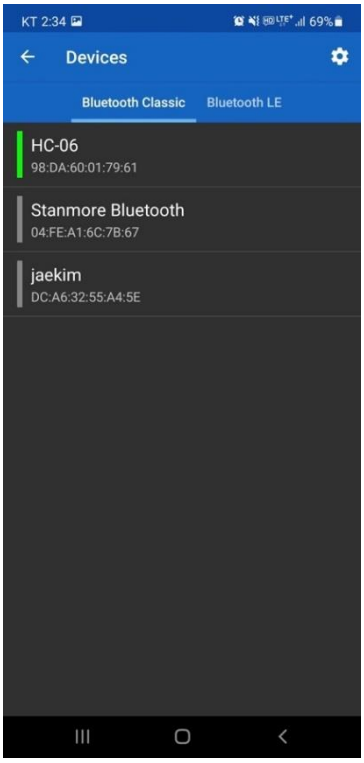
OBD 모듈에 임의의 코드를 입력 후, 정상적으로 작동하는지 시리얼 모니터를 통해 확인했다.
자동차 속도 받아오는 코드를 입력 후, 정상적으로 작동하는지 자동차를 통해 확인했다.

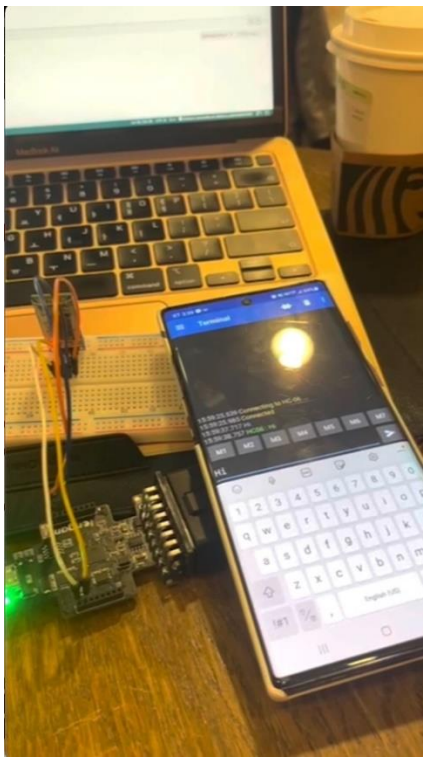




3-2. 블루투스 모듈 이상 유무 확인

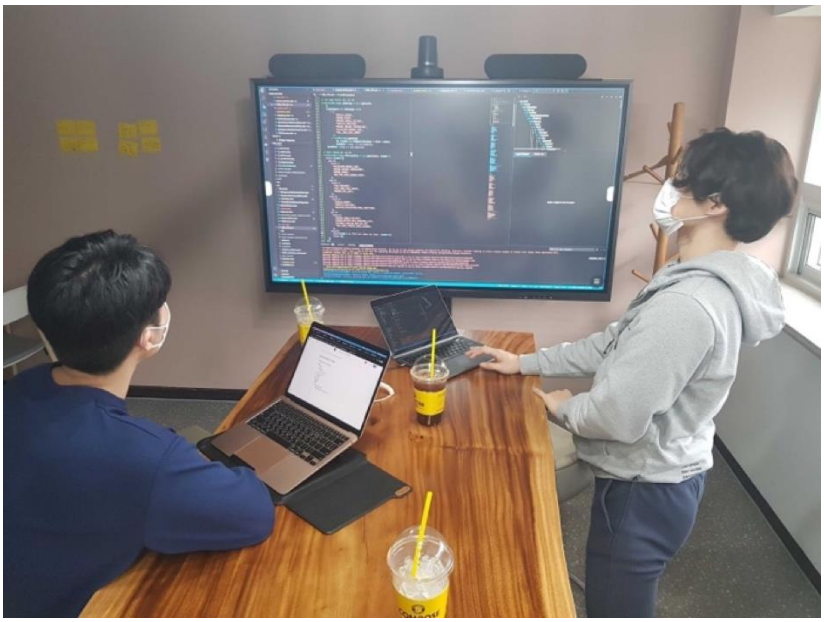
OBD 와 HC06(블루투스 모듈) 연결 후, 블루투스 테스트 앱을 통해 PC 와 스마트폰이 블루투스 통신을 정상적으로 하는지 시리얼 모니터로 확인했다.



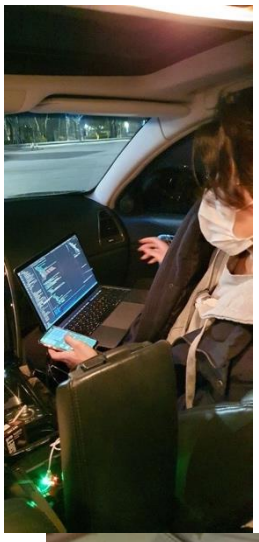


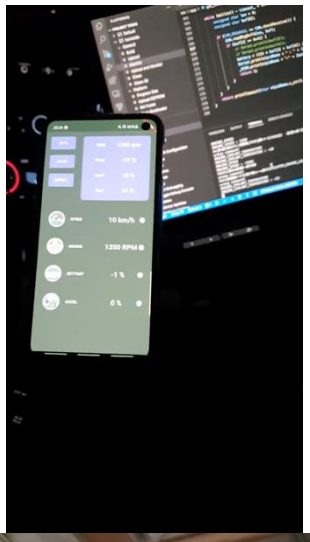
3-3. 앱 UI/UX 및 기능 순차적으로 구현

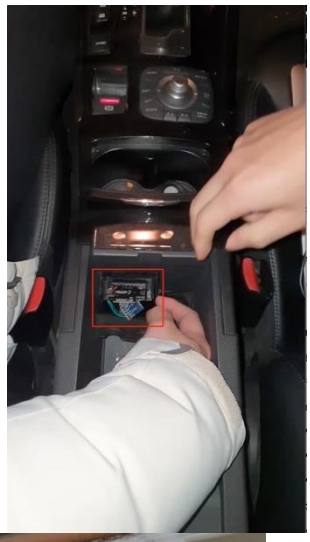
코드 리뷰를 하며 앱 개발을 진행했다. 순서는 UI/UX, 일반인/전문가 모드, 차량 진단, HUD 로 구현했다.



3-4. 테스트 진행



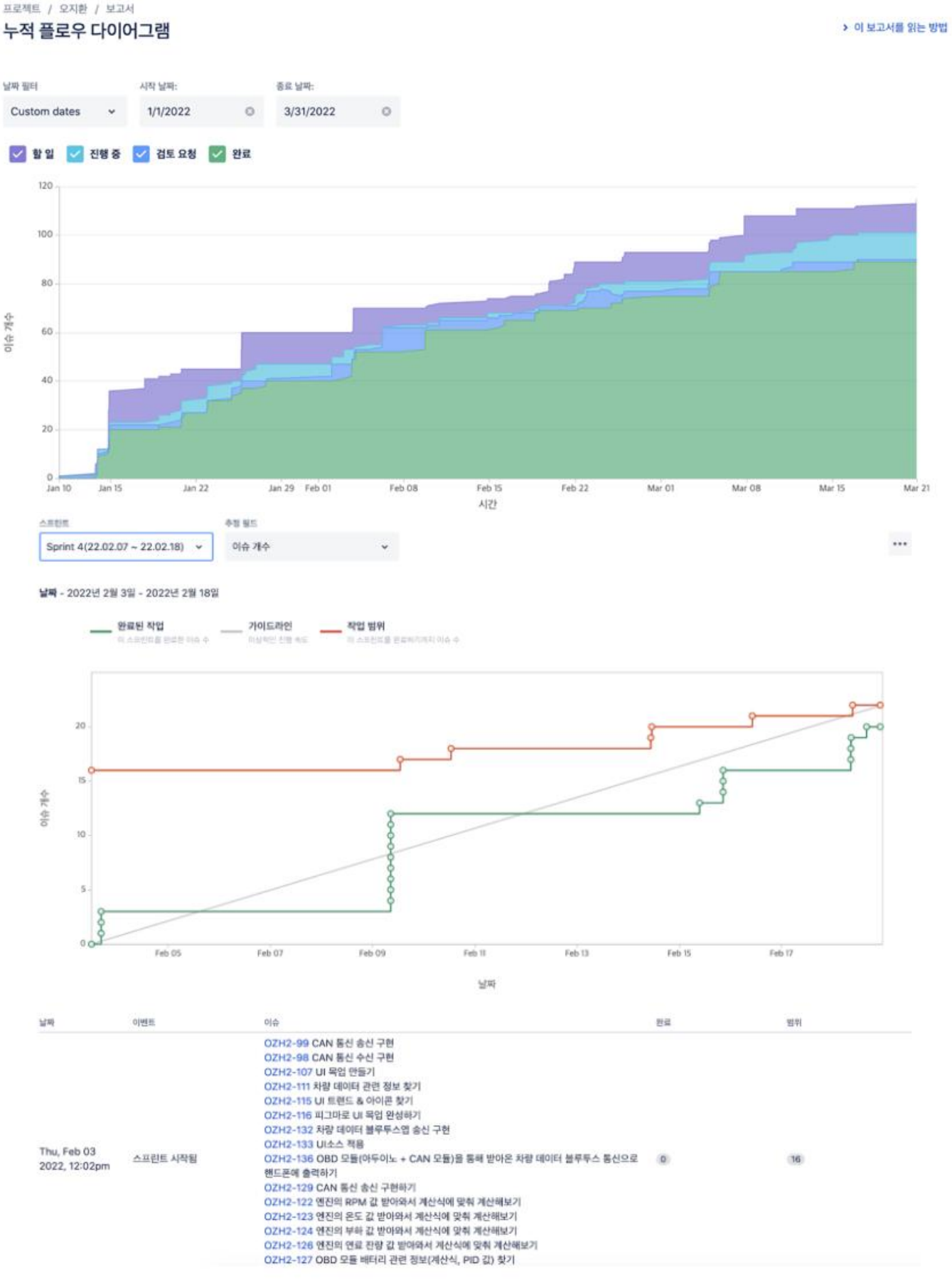




3-5. 프로젝트 진행

로드맵을 기반으로 스프린트를 진행했다. 스프린트 관리와 각종 문서는 Jira 와 Confluence 로 관리했다.

아래 이미지는 왼쪽부터 스프린트 전체에 대한 누적플로우차트와 4 번째 스프린트 진행 상황을 보여주는 번업 차트이다.



IV. 사용한 제품 리스트

제품명	디바이스마트 상품 번호
OBD-II CAN Bus GPS Development Kit	11983985
C 타입 To C 타입 케이블	12228066
블루투스 직렬포트 모듈 HC-06	1278220
CAN Bus 송수신 모듈 SZH-EKBG-030	1327552
ELM327	-

V. 참고 문헌

1. <자동차정비, 수리불량 부당수리비 청구 등 소비자불만 많아> 한국소비자원
https://www.kca.go.kr/home/sub.do?menukey=4005&mode=view&no=1001831095&searchKeyword=%EC%9E%90%EB%8F%99%EC%B0%A8

2. <설 장거리 운행 전 차량 점검, 선택 아닌 필수!> 불스원
https://bullsone.com/media/press/546?searchType=ALL&brandCode=ALL&searchString=%EC%B0%A8%EB%9F%89%20%EC%A0%90%EA%B2%80%20%EC%8B%A4%ED%83%9C

3. <불스원 설문조사, 운전자 10 명 중 8 명 "귀성 전 차량 점검 꼭 한다"> 불스원
https://bullsone.com/media/press/606?searchType=ALL&brandCode=ALL&searchString=%EC%B0%A8%EB%9F%89%20%EC%A0%90%EA%B2%80%20%EC%8B%A4%ED%83%9C

4. <코로나 19, 2020 년 판 '마이카 열풍' 기폭제?> 이코노믹리뷰
http://www.econovill.com/news/articleView.html?idxno=387804

5. <[모빌리티 인사이트] 이제 고개 들고 운전하세요, 헤드업디스플레이> IT 동아
https://it.donga.com/32328/

6. <'안전속도 5030' 탄력 운영>
http://www.enbnews.org/news/articleView.html?idxno=28884

7. <OBD 차량 진단 정보를 위한 IoT 장치 구현> 이성희, 이성형, 이상문, 황승훈
https://www.koreascience.or.kr/article/JAKO201608967045609.pdf

8. <CAN 통신을 이용한 자동차 엔진 정보 표시 장치> 박양재
https://www.koreascience.or.kr/article/JAKO201909055503007.pdf

9. <자동차용 ECU 다중 진단 시스템 및 방법> 권해윤
https://patentimages.storage.googleapis.com/a7/b2/e1/cadd7197f9f19a/KR101491260B1.pdf

10. <CAN 시스템의 시작과 진단> 임베디드시스템 코리아
http://www.eskorea.net/html/data/technique/IxCancheck.pdf

11. <CAN 통신 사용> RealSYS
http://realsys.co.kr/data/arm/10.CAN%ED%86%B5%EC%8B%A0%20%EC%82%AC%EC%9A%A9.pdf

12. <자동차 전자제어시스템(EUC)에서 생체인증을 이용한 인증기법> 이광재, 이근호
https://www.koreascience.or.kr/article/CFKO201335553771840.pdf

13. <CAN Protocol 을 이용한 CAN 통신 시스템 설계 및 구현> 이서경, 이재용, 김동현, 최광주, 정재일
https://scienceon.kisti.re.kr/commons/util/originalView.do?cn=CFKO200634741466208&oCn=NPAP08116964&dbt=CFKO&journal=NPRO00290005

14. <SPI 통신을 이용한 MMC 시스템의 Power Module DC 센싱 방법> 이종학, 신예슬, 김준구, 권병기
https://scienceon.kisti.re.kr/commons/util/originalView.do?cn=CFKO201432853253372&oCn=NPAP12691242&dbt=CFKO&journal=NPRO00377615

15. <차량 내부 통신을 위한 고속 CAN-FD 컨트롤러> 최도영, 윤영현, 오정환, 이승은
https://soc.seoultech.ac.kr/pdf/2019_JIEIE.pdf
16. <OBD-II PIDs> 위키 백과
https://en.wikipedia.org/wiki/OBD-II_PIDs
17. <CAN BUS> 위키 백과
https://en.wikipedia.org/wiki/CAN_bus
18. <SPI BUS> 위키 백과
https://ko.wikipedia.org/wiki/%EC%A7%81%EB%A0%AC_%EC%A3%BC%EB%B3%80%EA%B8%B0%EA%B8%B0_%EC%9D%B8%ED%84%B0%ED%8E%98%EC%9D%B4%EC%8A%A4_%EB%B2%84%EC%8A%A4
19. <OBD-II CAN 관련 표준 문서>
https://uglytree.tistory.com/51?category=283810
20. <OBD-II 송수신 함수>
https://cafe.naver.com/ittec/104
21. <CAN bus Databases by Car Vendors>
https://github.com/iDoka/awesome-automotive-can-id
22. <CAN 통신으로 OBD 구현하기>
https://sambumts.tistory.com/392
23. <CAN 통신을 통한 차량 점검 방법(HW)>
https://m.blog.naver.com/kuberabc/221861947082
24. <CAN 통신 구현(SW)>
https://makeutil.tistory.com/93

VI. 소스 코드

5-1. 아두이노

main.ino

```
#include "header.hpp"

SoftwareSerial HC06(HC06_RX, HC06_TX);
std::vector<Pair> pid_list;
MCP_CAN CAN = MCP_CAN(SPI_CS_PIN);
OBDPower obd = OBDPower(A3);

/*****************************************************************************************
** Function Name : setup
** Description         : call me when the program starts
*****************************************************************************************/
void setup() {
        SerialInit();
        obd.PowerOn();
        CANInit();
        setMaskFilt();
        BluetoothInit(HC06);
        initPidList(pid_list);
}

/*****************************************************************************************
** Function Name : loop
** Description         : call continuously until the program is over
*****************************************************************************************/
void loop() {
        if (HC06.available() > 0) {
                String request = HC06.readStringUntil('₩n');
                // std::string str = request.c_str();
                Serial.println(request);
                // findPid(pid_list, request.c_str(), HC06);
        }
}
```

CAN_protocol.cpp

```
#include "header.hpp"
#include <cstdlib>

/*****************************************************************************************
** Function Name : sendPid
** Description         : request data from the pid
*****************************************************************************************/
void sendPid(unsigned char pid) {
        unsigned char tmp[8] = {0x02, 0x01, pid, 0, 0, 0, 0, 0};
        CAN.sendMsgBuf(CAN_ID_PID, 0, 8, tmp);
}
```

```
/*********************************************************************************************
** Function Name : printTimeout
** Description         : print timeout message
*********************************************************************************************/
bool printTimeout(char *pid, SoftwareSerial &_HC06) {
        String _pid = pid;

        Serial.println(_pid + " : " + "Timeout");
        _HC06.println(_pid + ":" + "Timeout");
        return 0;
}
```

get_info.cpp

```
#include "header.hpp"

/*********************************************************************************************
** Function Name : getEngineRPM
** Description         : get engine RPM
*********************************************************************************************/
bool getEngineRPM(SoftwareSerial &_HC06) {
        String pidName = "ENGINE_SPEED";

        sendPid(ENGINE_SPEED);
        unsigned long timeout_ = millis();

        while (millis() - timeout_ < 1000) {
                unsigned char len = 0;
                unsigned char buf[8];
                int rpm;

                // check if get data
                if (CAN_MSGAVAIL == CAN.checkReceive()) {
                        CAN.readMsgBuf(&len, buf);
                        // 0x40 + 1(mode)
                        if (buf[1] == 0x41) {
                                // Serial.println(buf[3]);
                                // Serial.println(buf[4]);
                                rpm = buf[3] * 16 * 16;
                                rpm += buf[4];
                                Serial.println(pidName + " : " + (rpm / 4));
                                _HC06.println(pidName + ":" + (rpm / 4));

                                return 1;
                        }
                }
        }
        return printTimeout((char *)pidName.c_str(), _HC06);
```

```
}

/*************************************************************************************
** Function Name : getCoolantTemperature
** Description         : get coolant temperature
*************************************************************************************/
bool getCoolantTemperature(SoftwareSerial &_HC06) {
        String pidName = "ENGINE_COOLANT_TEMPERATURE";

        sendPid(ENGINE_COOLANT_TEMPERATURE);
        unsigned long timeout_ = millis();

        while (millis() - timeout_ < 1000) {
                unsigned char len = 0;
                unsigned char buf[8];
                int temp;

                if (CAN_MSGAVAIL == CAN.checkReceive()) {
                        CAN.readMsgBuf(&len, buf);
                        if (buf[1] == 0x41) {
                                // Serial.println(buf[3]);
                                temp = buf[3] - 40;
                                Serial.println(pidName + " : " + (temp));
                                _HC06.println(pidName + ":" + (temp));
                                return 1;
                        }
                }
        }
        return printTimeout((char *)pidName.c_str(), _HC06);
}

/*************************************************************************************
** Function Name : getEngineLoad
** Description         : get engine load(return percentage)
*************************************************************************************/
bool getEngineLoad(SoftwareSerial &_HC06) {
        String pidName = "CALCULATED_ENGINE_LOAD";

        sendPid(CALCULATED_ENGINE_LOAD);
        unsigned long timeout_ = millis();

        while (millis() - timeout_ < 1000) {
                unsigned char len = 0;
                unsigned char buf[8];
                int load;

                if (CAN_MSGAVAIL == CAN.checkReceive()) {
                        CAN.readMsgBuf(&len, buf);
                        if (buf[1] == 0x41) {
```

```
                                        // Serial.println(buf[3]);
                                        load = (buf[3] / 255.0) * 100.0;
                                        Serial.println(pidName + " : " + (load));
                                        _HC06.println(pidName + ":" + (load));
                                        return 1;
                                }
                        }
                }
                return printTimeout((char *)pidName.c_str(), _HC06);
}

/*********************************************************************************************
** Function Name : getFuelLevel
** Description          : get fuel level(return percentage)
*********************************************************************************************/
bool getFuelLevel(SoftwareSerial &_HC06) {
                String pidName = "FUEL_TANK_LEVEL_INPUT";

                sendPid(FUEL_TANK_LEVEL_INPUT);
                unsigned long timeout_ = millis();

                while (millis() - timeout_ < 1000) {
                        unsigned char len = 0;
                        unsigned char buf[8];
                        int fuel;

                        if (CAN_MSGAVAIL == CAN.checkReceive()) {
                                CAN.readMsgBuf(&len, buf);
                                if (buf[1] == 0x41) {
                                        // Serial.println(buf[3]);
                                        fuel = (buf[3] / 255.0) * 100.0;
                                        Serial.println(pidName + " : " + (fuel));
                                        _HC06.println(pidName + ":" + (fuel));
                                        return 1;
                                }
                        }
                }
                return printTimeout((char *)pidName.c_str(), _HC06);
}

/*********************************************************************************************
** Function Name : getSpeed
** Description          : get vehicle speed
*********************************************************************************************/
bool getSpeed(SoftwareSerial &_HC06) {
                String pidName = "VEHICLE_SPEED";

                sendPid(VEHICLE_SPEED);
                unsigned long timeout_ = millis();
```

```
	while (millis() - timeout_ < 1000) {
		unsigned char len = 0;
		unsigned char buf[8];

		if (CAN_MSGAVAIL == CAN.checkReceive()) {
			CAN.readMsgBuf(&len, buf);
			if (buf[1] == 0x41) {
				// Serial.println(buf[3]);
				Serial.println(pidName + " : " + (buf[3]));
				_HC06.println(pidName + ":" + (buf[3]));
				return 1;
			}
		}
	}
	return printTimeout((char *)pidName.c_str(), _HC06);
}

/*********************************************************************************************
** Function Name : getBattery
** Description	 : get battery
*********************************************************************************************/
bool getBattery(SoftwareSerial &_HC06) {
	String pidName = "BATTERY_CHARGE";

	sendPid(BATTERY_CHARGE);
	unsigned long timeout_ = millis();
	int battery;

	while (millis() - timeout_ < 1000) {
		unsigned char len = 0;
		unsigned char buf[8];

		if (CAN_MSGAVAIL == CAN.checkReceive()) {
			CAN.readMsgBuf(&len, buf);
			if (buf[1] == 0x41) {
				// Serial.println(buf[3]);
				// Serial.println(buf[4]);
				battery = (256 * buf[3] + buf[4]) / 1000.0;
				Serial.println(pidName + " : " + (battery));
				_HC06.println(pidName + ":" + (battery));
				return 1;
			}
		}
	}
	return printTimeout((char *)pidName.c_str(), _HC06);
}
```

OBD_power.cpp

```
#include "OBDPower.hpp"

/*********************************************************************************************************
** Function name:           OBDPower
** Descriptions:            set POWER_PIN
*********************************************************************************************************/
OBDPower::OBDPower(uint8_t power_pin):POWER_PIN(power_pin) {
}

/*********************************************************************************************************
** Function name:           PowerOn
** Descriptions:            power on
*********************************************************************************************************/
void OBDPower::PowerOn() {
        digitalWrite(POWER_PIN, HIGH);
}

/*********************************************************************************************************
** Function name:           PowerOff
** Descriptions:            power off
*********************************************************************************************************/
void OBDPower::PowerOff() {
        digitalWrite(POWER_PIN, LOW);
}
```

pid_list.cpp

```
#include "header.hpp"

/*********************************************************************************************************
** Function name:           inputPidList
** Descriptions:            input values in pid list
*********************************************************************************************************/
void inputPidList(std::vector<Pair> &pid_list, std::string name,
                                        unsigned int pid, bool *func(SoftwareSerial &_HC06)) {
        ASSERT(name.compare(""));
        ASSERT(pid != 0);
        ASSERT(func != 0);
        Pair pair(name, pid, func);
        pid_list.push_back(pair);
}

/*********************************************************************************************************
** Function name:           findPid
** Descriptions:            find pid at pid list
*********************************************************************************************************/
void findPid(std::vector<Pair> &pid_list, std::string name, SoftwareSerial &_HC06) {
        for (unsigned int i = 0; i < pid_list.size(); ++i) {
                std::string get_name = pid_list[i].getName();
```

```
                if (get_name.compare(name) == 0) {
                        return pid_list[i].startFunc(_HC06);
                }
        }
        Serial.println("wrong command!");
}
```

init.cpp

```
#include "header.hpp"

/*****************************************************************************************
** Function Name : SerialInit
** Description         : serial monitor init
*****************************************************************************************/
void SerialInit() {
        Serial.begin(SERIAL_SPEED);
        // while (!Serial)
        //          delay(1000);
        Serial.println("Serial Init ok!");
        Serial.println("---------------------------------");
}

/*****************************************************************************************
** Function Name : BluetoothInit
** Description         : Bluetooth App monitor init
*****************************************************************************************/
void BluetoothInit(SoftwareSerial &_HC06) {
        _HC06.begin(BLUETOOTH_SPEED);
}

/*****************************************************************************************
** Function Name : CANInit
** Description         : CAN init : buad rate = 500k
*****************************************************************************************/
void CANInit() {
        while (CAN_OK != CAN.begin(CAN_500KBPS)) {
                Serial.println("ERROR : CAN Init failed.");
                Serial.println("Retry...");
                Serial.println("---------------------------------");
                delay(1000);
        }
        Serial.println("CAN Init ok!");
        Serial.println("---------------------------------");
}

/*****************************************************************************************
** Function Name : setMaskFilt
```

```
** Description         : set mask and filt to get data from broadcast address(0x7FC)
*********************************************************************************************/
void setMaskFilt() {
        CAN.init_Mask(0, 0, 0x7FC);
        CAN.init_Mask(1, 0, 0x7FC);

        CAN.init_Filt(0, 0, 0x7E8);
        CAN.init_Filt(1, 0, 0x7E8);
        CAN.init_Filt(2, 0, 0x7E8);
        CAN.init_Filt(3, 0, 0x7E8);
        CAN.init_Filt(4, 0, 0x7E8);
        CAN.init_Filt(5, 0, 0x7E8);
        Serial.println("---------------------------------");
}

/*********************************************************************************************
** Function Name : initPidList
** Description         : set pid list
*********************************************************************************************/
void initPidList(std::vector<Pair> &pid_list) {
        std::string str = "ENGINE_SPEED";
        inputPidList(pid_list, str, OBDPid::ENGINE_SPEED, getEngineRPM);

        str = "ENGINE_COOLANT_TEMPERATURE";
        inputPidList(pid_list, str, OBDPid::ENGINE_COOLANT_TEMPERATURE, getCoolantTemperature);

        str = "CALCULATED_ENGINE_LOAD";
        inputPidList(pid_list, str, OBDPid::CALCULATED_ENGINE_LOAD, getEngineLoad);

        str = "FUEL_TANK_LEVEL_INPUT";
        inputPidList(pid_list, str, OBDPid::FUEL_TANK_LEVEL_INPUT, getFuelLevel);

        str = "VEHICLE_SPEED";
        inputPidList(pid_list, str, OBDPid::VEHICLE_SPEED, getSpeed);

        str = "BATTERY";
        inputPidList(pid_list, str, 164, getBattery);

        // check right name
        for (unsigned int i = 0; i < pid_list.size(); ++i)
                std::cout << pid_list[i].getName() <<   std::endl;
        Serial.println("Pid List Init ok!");
        Serial.println("---------------------------------");
}
```

header.hpp

```
#ifndef HEADER_HPP
#define HEADER_HPP
```

```
#include <stdio.h>
#include <Arduino.h>
#include <SPI.h>
#include <SoftwareSerial.h>
#include "ArduinoSTL.h"
#include "mcp_can.hpp"
#include "OBDPower.hpp"
#include "OBD_PID.hpp"
#include "Pair.hpp"

#define SERIAL_SPEED                9600
#define BLUETOOTH_SPEED             9600
#define SPI_CS_PIN                  9
#define CAN_ID_PID                  0x7DF
#define HC06_TX                     11
#define HC06_RX                     10
#define ASSERT(a)                   while (!a) ;

MCP_CAN CAN(SPI_CS_PIN);
OBDPower obd(A3);

/*
** init.cpp
*/
void SerialInit();
void CANInit();
void setMaskFilt();
void BluetoothInit(SoftwareSerial &_HC06);
void initPidList(std::vector<Pair> &pid_list);

/*
** get_info.cpp
*/
bool getEngineRPM(SoftwareSerial &_HC06);
bool getCoolantTemperature(SoftwareSerial &_HC06);
bool getEngineLoad(SoftwareSerial &_HC06);
bool getFuelLevel(SoftwareSerial &_HC06);
bool getSpeed(SoftwareSerial &_HC06);
bool getBattery(SoftwareSerial &_HC06);

/*
** CAN_protocol.cpp
*/
void sendPid(unsigned char pid);
bool printTimeout(char *pid, SoftwareSerial &_HC06) ;

/*
** pid_list.cpp
*/
```

```
void inputPidList(std::vector<Pair> &pid_list, std::string name, unsigned int pid,
                                        bool *func(SoftwareSerial &_HC06));
void findPid(std::vector<Pair> &pid_list, std::string name, SoftwareSerial &_HC06);

#endif
```

OBD_PID.hpp

```
#ifndef OBD_PID_HPP
#define OBD_PID_HPP

enum OBDPid
{
        /*
        ** 0x00 ~ 0x0F
        */
        PIDS_SUPPORTED_01_20,
        MONITOR_STATUS_SINXE_DTCS_CLEARED,
        FREEZE_DTC,
        FUEL_SYSTEM_STATUS,
        CALCULATED_ENGINE_LOAD,
        ENGINE_COOLANT_TEMPERATURE,
        SHORT_TERM_FUEL_TRIM_BANK_1,
        LONG_TERM_FUEL_TRIM_BANK_1,
        SHORT_TERM_FUEL_TRIM_BANK_2,
        LONG_TERM_FUEL_TRIM_BANK_2,
        FUEL_PRESSURE,
        INTAKE_MANIFOLD_ABSOLUTE_PRESSURE,
        ENGINE_SPEED,
        VEHICLE_SPEED,
        TIMING_ADVANCE,
        INTAKE_AIR_TEMPERATURE,

        /*
        ** 0x10 ~ 0x1F
        */
        MAF_AIR_FLOW_RATE,
        THROTTLE_POSITION,
        COMMANDED_SECONDARY_AIR_STATUS,
        OXYGEN_SENSORS_PRESENT_IN_2_BANKS,
        OXYGEN_SENSOR_1,
        OXYGEN_SENSOR_2,
        OXYGEN_SENSOR_3,
        OXYGEN_SENSOR_4,
        OXYGEN_SENSOR_5,
        OXYGEN_SENSOR_6,
        OXYGEN_SENSOR_7,
        OXYGEN_SENSOR_8,
        OBD_STANDARDS_THIS_VEHICLE_CONFORMS_TO,
```

```
    OXYGEN_SENSORS_PRESENT_IN_4_BANKS,
    AUXILIARY_INPUT_STATUS,
    RUN_TIME_SINCE_ENGINE_START,

    /*
    ** 0x20 ~ 0x2F
    */
    PIDS_SUPPORT_21_40,
    DISTANCE_TRAVELED_WITH_MIL_ON,
    FUEL_RAIL_PRESSURE,
    FUEL_RAIL_GAUGE_PRESSURE,
    OXYGEN_SENSOR_1_VOLTAGE,
    OXYGEN_SENSOR_2_VOLTAGE,
    OXYGEN_SENSOR_3_VOLTAGE,
    OXYGEN_SENSOR_4_VOLTAGE,
    OXYGEN_SENSOR_5_VOLTAGE,
    OXYGEN_SENSOR_6_VOLTAGE,
    OXYGEN_SENSOR_7_VOLTAGE,
    OXYGEN_SENSOR_8_VOLTAGE,
    COMMANDED_EGR,
    EGR_ERROR,
    COMMANDED_EVAPORATIVE_PURGE,
    FUEL_TANK_LEVEL_INPUT,

    /*
    ** 0x30 ~ 0x3F
    */
    WARM_UPS_SINCE_CODES_CLEARED,
    DISTANCE_TRAVELED_SINCE_CODES_CLEARED,
    EVAP_SYSTEM_VAPOR_PRESSURE,
    ABSOLULTE_BAROMETRIC_PRESSURE,
    OXYGEN_SENSOR_1_CURRENT,
    OXYGEN_SENSOR_2_CURRENT,
    OXYGEN_SENSOR_3_CURRENT,
    OXYGEN_SENSOR_4_CURRENT,
    OXYGEN_SENSOR_5_CURRENT,
    OXYGEN_SENSOR_6_CURRENT,
    OXYGEN_SENSOR_7_CURRENT,
    OXYGEN_SENSOR_8_CURRENT,
    CATALYST_TEMPERATURE_BANK_1_SENSOR_1,
    CATALYST_TEMPERATURE_BANK_2_SENSOR_1,
    CATALYST_TEMPERATURE_BANK_1_SENSOR_2,
    CATALYST_TEMPERATURE_BANK_2_SENSOR_2,

    /*
    ** 0x40 ~ 0x4F
    */
    PIDS_SUPPORT_41_60,
    MONITOR_STATUS_THIS_DRIVE_CYCLE,
```

```
        CONTROL_MODULE_VOLTAGE,
        ABSOLUTE_LOAD_VALUE,
        COMMANDED_FUEL_AIR_EQUIVALENCE_RATE,
        RELATIVE_THROTTLE_POSITION,
        AMBIENT_AIR_TEMPERATURE,
        ABSOLUTE_THROTTLE_POSITION_B,
        ABSOLUTE_THROTTLE_POSITION_C,
        ABSOLUTE_THROTTLE_POSITION_D,
        ABSOLUTE_THROTTLE_POSITION_E,
        ABSOLUTE_THROTTLE_POSITION_F,
        COMMANDED_THROTTLE_ACTUATOR,
        TIME_RUN_WITH_MIL_ON,
        TIME_SINCE_TROUBLE_CODES_CLEARED,
        MAXIMUM_VALUE_FOR_FUEL_AIR_EQUIVALENCE_RATIO,

        /*
        ** 0x50 ~ 0X5F
        */
        MAXIMUM_VALUE_FOR_AIR_FLOW_RATE_FROM_MASS_AIR_FLOW_SENSOR,
        FUEL_TYPE,
        ETHANOL_FUEL,
        ABSOLUTE_EVAP_SYSTEM_VAPOR_PRESSURE,
        EVAP_SYSTER_VAPOR_PRESSURE,
        SHORT_TERM_SECONDARY_OXYGEN_SENSOR_TRIM_1_3,
        LONG_TERM_SECONDARY_OXYGEN_SENSOR_TRIM_1_3,
        SHORT_TERM_SECONDARY_OXYGEN_SENSOR_TRIM_2_4,
        LONG_TERM_SECONDARY_OXYGEN_SENSOR_TRIM_2_4,
        FUEL_RAIL_ABSOLUTE_PRESSURE,
        RELATIVE_ACCELERATOR_PEDAL_POSITTION,
        HYBRID_BATTERY_PACK_REMAINING_LIFE,
        ENGINE_OIL_TEMPERATURE,
        FUEL_INJECTION_TIMING,
        ENGINE_FUEL_RATE,
        EMISSION_REQUIREMENT_TO_WHICH_VEHICLE_IS_DESIGNED,

        /*
        ** 0x60 ~ 0x6F
        */
        BATTERY_CHARGE = 164
};

#endif
```

5-2. 애플리케이션 코드

main.dart

코드의 시작부

```
import 'package:flutter/material.dart';
import 'homeListView.dart';
```

```
void main() => runApp(const Chada());

class Chada extends StatelessWidget {
 const Chada({Key key}) : super(key: key);
 @override Widget build(BuildContext context) {
 return MaterialApp(
 title: 'ChaDa',
 theme: ThemeData( colorScheme: ColorScheme.dark(),
 primarySwatch: Colors.indigo,
 ), home: MyPage(),
 ); } }
```

homeListView.dart

앱의 첫 번째 화면의 구성 및 기능

```
class listItem {
// >> 앱 화면의 레이아웃 구성을 위한 클래스
String imagePath;
String title;
String PIDname;
listItem(this.imagePath, this.title, this.PIDname);
}

class MyPage
// >> _buildBody 를 생성

class _buildBody
// >> _buildBody의 상태를 생성
class _buildBodyState
// >> 앱의 첫 화면을 생성
carInfo info = carInfo();
bool showMainMenu = true;
bool showBottomMenu = false;
bool selectEasyMode = true;

void initState()
// >> 화면 생성 시 진행할 초기화 함수, 여기서는 초당 2회 화면 갱신

Widget build(BuildContext context)
// >> 주 화면과 블루투스 설정 창의 레이아웃을 생성
// >> 설정창이 화면 위로 겹쳐서 올라오고, 배경이 불투명해지도록 함

Widget mainUpper(BuildContext context)
// >> 주 화면 윗 부분의 레이아웃을 생성
// >> 좌측의 버튼 4개와, 우측의 PID그룹 창을 생성

Widget mainMenuButtons(BuildContext context)
// 화면 상단 좌측의 버튼 4개를 생성 Widget menuButtons( {String title, ButtonStyle style, void Function() onPressed})
// 버튼의 모양을 만들어줌
```

```
Widget mainMenuPIDList(BuildContext context)
// 화면 우측 상단, PID그룹 리스트의 위치를 지정

Widget mainSimpleInfo(BuildContext context)
// PID그룹 리스트를 일반인, 전문가 모드 중 현재 모드에 맞게 생성함

Widget pidGroupCardEasy(int index)
// 일반인 모드에서 PID그룹 리스트의 항목을 생성함

Widget pidGroupCard(int pidGroupIndex)
// 전문가 모드에서 PID그룹 리스트의 항목을 생성함

Widget mainList()
// 주 화면 하단의 실시간 PID 정보 표시 리스트를 생성

Widget itemCard(listItem item)
// 실시간 PID 정보 표시 리스트의 항목 생성

listItem toListItem(String pid)
// PID에 맞는 리스트 카드를 반환
```