vm_serial_port.dart

소스 코드

전체 코드

```dart
import 'dart:async';
import 'dart:typed_data';

import 'package:buyoil/common/app_commands.dart';
import 'package:buyoil/common/utils/show_toast.dart';
import 'package:buyoil/common/utils/toast/toast_provider.dart';
import 'package:buyoil/model/ui_state_usb_port.dart';
import 'package:buyoil/router.dart';
import 'package:buyoil/viewmodel/vm_step1.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:fluttertoast/fluttertoast.dart';
import 'package:riverpod_annotation/riverpod_annotation.dart';
import 'package:usb_serial/usb_serial.dart';

import '../common/app_strings.dart';
import '../config.dart';

part 'vm_serial_port.g.dart';


@Riverpod(keepAlive: true)
class SerialPortVM extends _$SerialPortVM {
  StreamSubscription<Uint8List>? _subscription;

  final StringBuffer _receiveBuffer = StringBuffer();
  static const String _terminator = '#';

  // final RegExp formatRegex = RegExp(r'^(\[ANS\])?O(\d+(\.\d+)?)W(\d+(\.\d+)?)E$');
  final RegExp formatRegex = RegExp(r'^(?:\[ANS\])?O\d+(\.\d+)?W\d+(\.\d+)?E#?$');
  int _connectionAttemptIndex = 0;
  Timer? _inactivityTimer; // <--- 1. 비활성 상태 감지를 위한 타이머 추가

  int _reconnectAttempts = 0;
  final int _maxReconnectAttempts = 10; // 최대 10번까지만 시도
  Timer? _reconnectTimer;

  @override
  UIStateUsbPort build() {
    print("SerialPortVM: build();");
    // ViewModel이 처음 생성될 때 재시도 카운터 초기화
    _reconnectAttempts = 0;

    // Riverpod 2.0에서는 @dispose 어노테이션을 사용하거나,
```

```
    // ref.onDispose를 사용하여 리소스를 정리합니다.
    ref.onDispose(() {
      _subscription?.cancel();
      try {
        state.port?.close();
      } catch (e) {}
      _inactivityTimer?.cancel();
      _reconnectTimer?.cancel(); // 재연결 타이머도 취소
      print("SerialPortVM disposed.");
    });

    _init();
    _resetInactivityTimer();
    return UIStateUsbPort.init(availablePorts: []);
  }

  // <--- 2. 타이머를 시작하고 재설정하는 헬퍼 메서드 추가 ---
  void _resetInactivityTimer() {
    _inactivityTimer?.cancel(); // 기존 타이머가 있다면 취소
    _inactivityTimer = Timer.periodic(const Duration(seconds: kDebugMode ? 120 : 120),
(timer) {
      // 120초 동안 아무런 쓰기 작업이 없으면 SLEEP 명령어 전송
      showScafold(AppStrings.trySleep);
      writeToPort(PORT_COMMANDS.sleep);
    });
  }

  _init() async {
    connectPort();
  }

  Future<void> connectPort({BuildContext? context}) async {
    // 1. 기존 연결 및 버퍼 정리
    _subscription?.cancel(); // 이전 구독이 있다면 취소
    try {
      state.port?.close(); // 이전 포트가 있다면 닫기
    } catch(e) {}
    _receiveBuffer.clear();

    // 재연결 시도가 아닐 경우에만 init() 상태로 변경 (연결 끊김 메시지 유지를 위해)
    if (_reconnectAttempts == 0) {
      state = UIStateUsbPort.init();
    }

    List<UsbDevice> devices = await UsbSerial.listDevices();

    // // // 임시
    // if (Config.instance.isDebugMode && devices.isEmpty) {
    //   final List<UsbDevice> demoDevices = List.generate(
    //     2,
    //     (i) => UsbDevice(
    //       // 실제 생성자 매개변수 순서와 타입에 맞게 수정
    //       '/dev/ttyACM$i', // macOS/Linux 스타일의 장치 이름
```

```
    //       1234 + i,
    //       5678 + i,
    //       'Select This Debug Device For Test',
    //       'DemoCorp',
    //       1000 + i,
    //       'DEMO-SN-00${i + 1}',
    //       1,
    //     ),
    //   );
    //   devices.addAll(demoDevices);
    //   print("Added 5 demo devices for debugging.");
    // }

    state = state.copyWith(availablePorts: devices);

    String printString = "";
    devices.forEach((dv) {
      printString += dv.deviceName;
      printString += "\n";
    });
    showScafold("devices[${devices.length}\n(${printString})");

    if (devices.isEmpty) {
      showScafold(AppStrings.noConnectableDevicesFound);
      state = UIStateUsbPort.error(errorMsg: AppStrings.noConnectableDevicesFound);
      return;
    }

    UsbDevice? deviceToConnect;

    try {
      deviceToConnect = devices.firstWhere(
            (d) =>
        (d.productName?.toLowerCase().contains("stm32") ?? false) ||
            (d.productName?.toLowerCase().contains("usb serial") ?? false),
      );
      showScafold("Auto-selecting device: ${deviceToConnect.productName}");
      print("Auto-selecting device: ${deviceToConnect.productName}");
    } catch (e) {
      // firstWhere가 장치를 찾지 못하면 에러를 발생시키므로, 여기서 null 처리
      deviceToConnect = null;
      print("No auto-selectable device found. Proceeding to manual selection.");
    }

    if (deviceToConnect == null) {
      if (devices.length > 1 && context != null && context.mounted) {
        // 장치가 여러 개일 경우 선택 팝업 표시
        deviceToConnect = await showDialog<UsbDevice>(
          context: context,
          builder: (BuildContext dialogContext) {
            return AlertDialog(
              title: const Text('Select Device'),
              content: SizedBox(
```

```
                width: 350,
                child: ListView.builder(
                  shrinkWrap: true,
                  itemCount: devices.length,
                  itemBuilder: (BuildContext _, int index) {
                    final device = devices[index];
                    return ListTile(
                      visualDensity: VisualDensity.compact,
                      contentPadding: const EdgeInsets.symmetric(horizontal: 16.0, vertical: 0),
                      title: Text("${index + 1}/${devices.length} ${device.productName ?? 'Unknown Device'}"),
                      subtitle: Text('VID: ${device.vid}, PID: ${device.pid}'),
                      onTap: () {
                        Navigator.of(dialogContext).pop(device);
                      },
                    );
                  },
                ),
              ),
              actions: <Widget>[
                TextButton(
                  child: const Text('Cancel'),
                  onPressed: () {
                    Navigator.of(dialogContext).pop();
                  },
                ),
              ],
            );
          },
        );

        if (deviceToConnect == null) {
          showScafold(AppStrings.deviceSelectionCanceled);
          state = UIStateUsbPort.error(errorMsg: AppStrings.deviceSelectionCanceled);
          return;
        }
      } else {
        // 장치가 하나뿐인 경우 자동으로 첫 번째 장치 선택
        deviceToConnect = devices.first;
      }
    }

    showScafold("${AppStrings.tryToConnect}: ${deviceToConnect.productName ?? deviceToConnect.deviceName}");
    /// 디버그 기기 선택 시, 별도 동작 진행
    if(Config.instance.isDebugMode && deviceToConnect.productName == "Select This Debug Device For Test") {
      _connectionAttemptIndex = 0;
      UsbPort port = UsbPort("test port");
      // --- (포트 설정 및 listen 로직은 기존과 동일) ---
      await port.setDTR(true);
      await port.setRTS(true);
```

```
      await port.setPortParameters(115200, UsbPort.DATABITS_8,
          UsbPort.STOPBITS_1, UsbPort.PARITY_NONE);

      _subscription = port.inputStream!.listen((Uint8List data) {
        if (_reconnectAttempts > 0) {
          showScafold("Device reconnected successfully!");
          _reconnectAttempts = 0; // 성공했으므로 재시도 카운터 초기화
          _reconnectTimer?.cancel();
        }

        final receivedString = String.fromCharCodes(data);
        _receiveBuffer.write(receivedString);
        showScafold("Received chunk: $receivedString (Buffer:
${_receiveBuffer.toString()})");

        // 3. 버퍼에 종료 문자가 있는지 확인하고, 있으면 처리
        _processBuffer();
        },
        onError: (error) {
          print("Port stream error: $error. Attempting to reconnect...");
          showScafold("Connection error. Trying to reconnect...");
          state = UIStateUsbPort.error(errorMsg: "Connection error: $error");
          _handleDisconnection();
        },
          onDone: () {
            print("Port connection lost (onDone). Attempting to reconnect...");
            showScafold("Connection lost. Trying to reconnect...");
            state = UIStateUsbPort.error(errorMsg: "Connection lost. Retrying...");
            _handleDisconnection();
      });

      state = UIStateUsbPort.connected(port: port, connectedDevice: deviceToConnect);
      final connectedDeviceName = deviceToConnect.productName ??
deviceToConnect.deviceName;
      final toastMessage = "Connection Succeeded: $connectedDeviceName
(VID:${deviceToConnect.vid}, PID:${deviceToConnect.pid})";

      showScafold(toastMessage);
      _resetInactivityTimer();
      return;
    }

    // 3. 포트 생성 및 연결 (기존 로직과 동일)
    UsbPort? port = await deviceToConnect.create();
    if (port == null) {
      state = UIStateUsbPort.error(errorMsg: AppStrings.cannotCreatePort);
      showScafold(AppStrings.cannotCreatePort);
      _handleDisconnection(); // 포트 생성 실패 시에도 재연결 시도
      return;
    }

    bool openResult = await port.open();
    if (!openResult) {
```

```
250       state = UIStateUsbPort.error(errorMsg: AppStrings.cannotOpenPort);
251       _handleDisconnection(); // 포트 열기 실패 시에도 재연결 시도
252       showScafold(AppStrings.cannotOpenPort);
253       return;
254     }
255
256     // 연결 성공 시 재시도 카운터 초기화
257     _connectionAttemptIndex = 0;
258     _reconnectAttempts = 0; // 재연결 성공했으므로 카운터 초기화
259     _reconnectTimer?.cancel();
260
261     // --- (포트 설정 및 listen 로직은 기존과 동일) ---
262     await port.setDTR(true);
263     await port.setRTS(true);
264     await port.setPortParameters(115200, UsbPort.DATABITS_8,
265         UsbPort.STOPBITS_1, UsbPort.PARITY_NONE);
266
267     _subscription = port.inputStream!.listen((Uint8List data) {
268       // 데이터 수신 시 재연결 상태였다면 완전한 성공으로 간주
269       if (_reconnectAttempts > 0 || _reconnectTimer?.isActive == true) {
270         showScafold("Device reconnected successfully!");
271         print("Device reconnected successfully!");
272         _reconnectAttempts = 0;
273         _reconnectTimer?.cancel();
274       }
275
276       final receivedString = String.fromCharCodes(data);
277       _receiveBuffer.write(receivedString);
278       showScafold("Received chunk: $receivedString (Buffer:
    ${_receiveBuffer.toString()})");
279       // 3. 버퍼에 종료 문자가 있는지 확인하고, 있으면 처리
280       _processBuffer();
281     }, onDone: () {
282       print("Port connection lost (onDone).");
283       state = UIStateUsbPort.error(errorMsg: "Connection lost. Retrying...");
284       _handleDisconnection();
285     },
286     onError: (error) {
287       print("Port stream error: $error.");
288       state = UIStateUsbPort.error(errorMsg: "Connection error: $error");
289       _handleDisconnection();
290     });
291     state = UIStateUsbPort.connected(port: port, connectedDevice: deviceToConnect);
292     final connectedDeviceName = deviceToConnect.productName ??
    deviceToConnect.deviceName;
293     final toastMessage = "Connection Succeeded: $connectedDeviceName
    (VID:${deviceToConnect.vid}, PID:${deviceToConnect.pid})";
294
295     showScafold(toastMessage);
296     _resetInactivityTimer(); // <--- 4. 연결 성공 시 타이머 시작
297   }
298
299   /// 연결이 끊겼을 때 호출되는 공통 핸들러
```

```dart
  void _handleDisconnection() {
    // 기존 리소스 정리
    _subscription?.cancel();
    _subscription = null;
    try {
      state.port?.close();
    } catch (e) {
      print("Error closing port during disconnection: $e");
    }

    // 재연결 시도 시작
    _attemptReconnect();
  }

  /// 재연결을 시도하는 함수
  void _attemptReconnect() {
    // 이미 재연결 타이머가 동작 중이면 중복 실행 방지
    if (_reconnectTimer?.isActive ?? false) return;

    if (_reconnectAttempts >= _maxReconnectAttempts) {
      print("Max reconnect attempts reached. Stopping reconnection.");
      showScafold("Failed to reconnect. Please check the connection.");
      state = UIStateUsbPort.error(
        errorMsg: "Failed to reconnect. Please check the cable and restart the app.",
      );
      _reconnectAttempts = 0; // 다음 수동 연결을 위해 초기화
      return;
    }

    _reconnectAttempts++;
    print("Attempting to reconnect... ($_reconnectAttempts/$_maxReconnectAttempts)");
    showScafold("Reconnection attempt $_reconnectAttempts...");

    // 3초 후에 `connectPort` 함수를 다시 호출하여 연결 시도
    _reconnectTimer = Timer(const Duration(seconds: 3), () {
      print("Executing reconnect via connectPort()");
      connectPort();
    });
  }

  /// 포트에 쓰기
  Future<void> writeToPort(PORT_COMMANDS command, {BuildContext? context}) async {
    print("writeToPort");
    // SLEEP 명령 자체는 타이머를 재설정하지 않도록 예외 처리
    if (command != PORT_COMMANDS.sleep) {
      _resetInactivityTimer(); // <--- 5. 쓰기 작업 시 타이머 재설정
    }

    if(Config.instance.isDebugMode) {
      state = state.copyWith(lastCommand: command);
      String packet = "${command.command}#";
      final dataToSend = Uint8List.fromList(packet.codeUnits);
      write(dataToSend);
```

```
      return;
    }

    if(state is UIStateUsbPortConnected) {
      state = state.copyWith(lastCommand: command);
      String packet = "${command.command}#";
      final dataToSend = Uint8List.fromList(packet.codeUnits);
      write(dataToSend);
    } else {
      if(context != null && context.mounted) {
        showToastMessage(context, "Port not connected");
      }
    }
  }

  Future<void> writeToPortPhone(String phoneCommand, {BuildContext? context}) async {
    _resetInactivityTimer(); // <--- 6. 쓰기 작업 시 타이머 재설정

    if(Config.instance.isDebugMode) {
      state = state.copyWith(lastCommand: PORT_COMMANDS.cmdPhone);
      print("[DEBUG] writeToPort: ${phoneCommand}");
      String packet = "$phoneCommand#";
      final dataToSend = Uint8List.fromList(packet.codeUnits);
      write(dataToSend);
    }

    print("[DEBUG] writeToPort: phone: $phoneCommand");
    if(state is UIStateUsbPortConnected) {
      state = state.copyWith(lastCommand: PORT_COMMANDS.cmdPhone);
      String packet = "$phoneCommand#";
      final dataToSend = Uint8List.fromList(packet.codeUnits);
      write(dataToSend);
    } else {
      if(context != null && context.mounted) {
        showToastMessage(context, "Port not connected");
      }
    }
  }

  // 포트에서 받은 값 처리
  // 직전 명령에 따라서 동작 별도 처리
  void listenByPort(String receivedString) {
    if(!receivedString.startsWith(AppCommands.prefixAnswer)) {
      showScafold("Prefix String is not [ANS]");
      return ;
    }
    showScafold("[${state.lastCommand}]listenByPort:
$receivedString#\n_receiveBuffer:${_receiveBuffer}");
    print("[DEBUG] lastCommand  : ${state.lastCommand}");
    print("[DEBUG] listenByPort : $receivedString");
    // 최초 화면에서
    if(state.lastCommand == PORT_COMMANDS.handshake) {
      // OK 받으면
```

```
405       if(receivedString == PORT_RESPONSES.ok.response) {
406         // 전화번호 입력 페이지 이동
407         ref.watch(routerProvider).goNamed(RouteGroup.Step1.name);
408       } else if(receivedString == PORT_RESPONSES.fail.response) {
409         // 아니면 기본 화면 유지
410         ref.watch(routerProvider).goNamed(RouteGroup.Splash.name);
411       } else if(receivedString == PORT_RESPONSES.full.response) {
412         ref.watch(routerProvider).goNamed(RouteGroup.Splash.name);
413         // 1. 5초간 팝업 띄우기
414         ref.read(toastProvider.notifier).showToast(AppStrings.fullContainer, duration:
    Duration(seconds: 5));
415       }
416     }
417
418     if (receivedString == PORT_RESPONSES.fail.response) {
419       state = state.copyWith(lastCommand: null);
420       // 2. 첫 페이지(Splash)로 이동
421       ref.watch(routerProvider).goNamed(RouteGroup.Splash.name);
422       // 1. 5초간 팝업 띄우기
423       ref.read(toastProvider.notifier).showToast(AppStrings.systemError, duration:
    Duration(seconds: 5));
424       return;
425     }
426
427     // [ANS]STM_SLEEP# 응답을 받으면
428     if (receivedString == PORT_RESPONSES.stmSleep.response) {
429       showScafold("Received STM_SLEEP. Going to Splash.");
430       // 스플래시 화면으로 이동
431       ref.watch(routerProvider).goNamed(RouteGroup.Splash.name);
432       // lastCommand를 초기화하여 추가 동작 방지
433       state = state.copyWith(lastCommand: null);
434       return; // 이 응답 처리는 여기서 종료
435     }
436
437     // 전화번호 send 응답이
438     if(state.lastCommand == PORT_COMMANDS.cmdPhone) {
439       // OK 받으면
440       if(receivedString == PORT_RESPONSES.ok.response) {
441         // 성공 시 실패 카운터 초기화
442         state = state.copyWith(resetFailCount: 0);
443
444         // page4(Step2) 오픈 화면 이동
445         ref.watch(routerProvider).goNamed(RouteGroup.Step2.name);
446       } else if(receivedString == PORT_RESPONSES.open.response) {
447         // 성공 시 실패 카운터 초기화
448         state = state.copyWith(resetFailCount: 0);
449         // Opening Door Screen 이동
450         ref.watch(routerProvider).goNamed(RouteGroup.OpeningDoor.name);
451       } else if(receivedString == PORT_RESPONSES.fail.response) {
452         // todo 변경사항 확인하기
453         ref.read(step1Provider.notifier).showErrorToast();
454       } else if(receivedString == PORT_RESPONSES.reject.response) {
455         final newFailCount = (state.resetFailCount ?? 0) + 1;
```

```
        state = state.copyWith(resetFailCount: newFailCount);
        if (newFailCount >= 5) {
          // 1. 5회 실패: 첫 페이지로 복귀
          ref.read(toastProvider.notifier).showToast(
              AppStrings.accessDenied,
              duration: Duration(seconds: 5));
          ref.watch(routerProvider).goNamed(RouteGroup.Splash.name);
          // 상태 초기화
          state = state.copyWith(lastCommand: null, resetFailCount: 0);
        } else {
          // 2. 5회 미만 실패: 팝업 메시지 표시 (페이지는 그대로 유지)
          ref.read(toastProvider.notifier).showToast(
              AppStrings.accessDenied,
              duration: Duration(seconds: 3)
          );
        }
        // todo 변경사항 확인하기
      } else if(receivedString == PORT_RESPONSES.notAuth.response) {
        // *return “NOTAUTH” -> Please shows the proper driver code
        // TODO
      } else if(receivedString == PORT_RESPONSES.driverTrue.response) {
        // send "[CMD]OPENB#" and shows driver's page.
        writeToPort(PORT_COMMANDS.openB).whenComplete(() {
          goToDriverPage();
        });
      } else if(receivedString == PORT_RESPONSES.driverFalse.response) {
        ref.read(toastProvider.notifier).showToast(AppStrings.notAuthorized);
        // send "[CMD]SLEEP#" and shows not authorized.
        writeToPort(PORT_COMMANDS.sleep);
      }
    }

    // Open 명령어
    if(state.lastCommand == PORT_COMMANDS.open) {
      // ok 응답 -> Close 화면 이동
      if(receivedString == PORT_RESPONSES.ok.response) {
        ref.watch(routerProvider).goNamed(RouteGroup.Step3.name);
      }
    }

    // 'postper'
    if (state.lastCommand == PORT_COMMANDS.postper) {
      if (receivedString == PORT_RESPONSES.ok.response) {
        // 1. postper에 대한 응답을 받으면 lastCommand를 초기화하여 재시도 타이머를 멈춤
        state = state.copyWith(lastCommand: null);
        // 1-1. 응답이 'OK'이면, 바로 'postData'를 전송
        print("Received OK for 'postper', now sending 'postData'.");
        showScafold("Received OK for 'postper', now sending 'postData'.");
        // poster -> postData 로 이어지므로 마지막 명령어 자체 업데이트
        state = state.copyWith(lastCommand: PORT_COMMANDS.postData);
        okayNextStep(); // postData를 보내는 전용 함수 호출
        return;
      }
```

```
    }

    if(state.lastCommand == PORT_COMMANDS.postData) {
      if (receivedString == PORT_RESPONSES.ok.response) {
        // 2. postData에 대한 응답을 받으면 lastCommand를 초기화하여 재시도 타이머를 멈춤
        state = state.copyWith(lastCommand: null);
        // 2-1. 응답이 'OK'이면 모든 과정이 성공했으므로 스플래시 화면으로 이동
        print("Received OK for 'postData'. Process complete.");
        showScafold("Received OK for 'postData'. Process complete.");
        ref.watch(routerProvider).goNamed(RouteGroup.Splash.name);
      }
    }

    // Close 명령어
    if(state.lastCommand == PORT_COMMANDS.close) {
      // O, W, E 포맷이면
      if(formatRegex.hasMatch(receivedString)) {
        state = state.copyWith(lastCommand: null);
        (double oil, double water) res =
AppCommands.returnOilWaterFormat(receivedString);

        // 초기화 후 Step4화면 재실행
        if(state.connectedDevice != null && state.port != null) {
          state = UIStateUsbPortConnected(
            availablePorts: state.availablePorts,
            connectedDevice: state.connectedDevice!,
            port: state.port!,
            lastCommand: null,
          );
        }

        ref.read(routerProvider).goNamed(RouteGroup.Step4.name, queryParameters: {
          "oil": "${res.$1}",
          "water": "${res.$2}",
        });
      }
    }

    if(state.lastCommand == PORT_COMMANDS.recheck) {
      // O, W, E 포맷이면
      if(formatRegex.hasMatch(receivedString)) {
        // 초기화 후 Step4화면 재실행
        state = UIStateUsbPortConnected(
          availablePorts: state.availablePorts,
          connectedDevice: state.connectedDevice,
          port: state.port,
          lastCommand: null,
        );

        (double oil, double water) res =
AppCommands.returnOilWaterFormat(receivedString);
        // 초기화 후 Step4화면 재실행
        print("recheck success: ${state.connectedDevice}/ ${state.port}");
```

```
560           ref.read(routerProvider).goNamed(RouteGroup.Step4.name, queryParameters: {
561             "oil": "${res.$1}",
562             "water": "${res.$2}",
563           });
564         }
565       }
566
567       // ok 명령 받은 경우 lastCommand 초기화
568       if(receivedString == PORT_RESPONSES.ok.response) {
569         if(state.lastCommand == PORT_COMMANDS.postData || state.lastCommand ==
    PORT_COMMANDS.postper || state.lastCommand == PORT_COMMANDS.recheck) {
570           return;
571         }
572         state = state.copyWith(lastCommand: null);
573       }
574     }
575
576     // 스플래시 화면 이동 및 close 명령
577     Future<void> goToSplash() async {
578       return await writeToPort(PORT_COMMANDS.close);
579     }
580
581     // 드라이버 페이지로 이동
582     void goToDriverPage() {
583       ref.read(routerProvider).goNamed(RouteGroup.Driver.name);
584     }
585
586     // open
587     Future<void> open() async {
588       // 30초 이내에 ok 안오면 재시도
589       Future.delayed(Duration(seconds: Config.instance.isDebugMode ? 5 : 30), () {
590         if(state.lastCommand == PORT_COMMANDS.open) {
591           open();
592         }
593       });
594
595       return writeToPort(PORT_COMMANDS.open);
596     }
597
598     Future<void> close() async {
599       // 30초 이내에 ok 안오면 재시도
600       Future.delayed(Duration(seconds: Config.instance.isDebugMode ? 5 : 30), () {
601         if(state.lastCommand == PORT_COMMANDS.close) {
602           close();
603         }
604       });
605
606       return writeToPort(PORT_COMMANDS.close);
607     }
608
609     Future<void> start() async {
610       return await writeToPort(PORT_COMMANDS.handshake);
611     }
```

```dart

  Future<void> okay() async {
    // 1. 'postper' 명령에 대한 재시도 로직 설정
    Future.delayed(Duration(seconds: Config.instance.isDebugMode ? 5 : 10), () {
      // 30초 후에도 lastCommand가 여전히 postper이면, 응답이 없는 것이므로 재시도
      if (state.lastCommand == PORT_COMMANDS.postper) {
        showScafold("No response for 'postper', retrying...");
        print("No response for 'postper', retrying...");
        okay(); // 'postper' 전송 재시도
      }
    });

    // 2. 'postper' 명령어 전송
    await writeToPort(PORT_COMMANDS.postper);

    // 3. 로딩 상태로 변경
    state = UIStateUsbPort.loading(
      availablePorts: state.availablePorts,
      connectedDevice: state.connectedDevice,
      port: state.port,
      lastCommand: state.lastCommand,
    );
  }

  /// 'okay' 프로세스의 두 번째 단계: 'postData' 전송 및 재시도 설정
  Future<void> okayNextStep() async {
    // 1. 'postData' 명령에 대한 타임아웃/재시도 로직 설정
    Future.delayed(Duration(seconds: Config.instance.isDebugMode ? 5 : 10), () {
      if (state.lastCommand == PORT_COMMANDS.postData) {
        showScafold("No response for 'postData', retrying...");
        print("No response for 'postData', retrying...");
        okayNextStep(); // 'postData' 전송 재시도
      }
    });

    // 2. 'postData' 명령어 전송
    await writeToPort(PORT_COMMANDS.postData);
  }

  Future<void> recheck() async {
    state = UIStateUsbPort.loading(
      availablePorts: state.availablePorts,
      connectedDevice: state.connectedDevice,
      port: state.port,
      lastCommand: PORT_COMMANDS.recheck,
    );
    return await writeToPort(PORT_COMMANDS.recheck);

  }

  Future<void> sendPhoneNumber(String phoneNumber) async {
    return await writeToPortPhone(PORT_COMMANDS.getValidPhoneCommand(phoneNumber));
  }
```

```dart

  /// [DEBUG] 특정 응답을 받은 것처럼 시뮬레이션
  void listenDebug(String data) {
    // 1. 디버그용 데이터를 실제 수신 버퍼에 추가합니다.
    _receiveBuffer.write(data);
    showScafold("Debug data injected: $data (Buffer: ${_receiveBuffer.toString()})");

    // 2. 실제 데이터가 들어왔을 때와 동일한 버퍼 처리 로직을 호출합니다.
    _processBuffer();
  }

  void writeDebug(String msg) {
    // 1. 디버그 모드가 아니거나, 포트가 연결되지 않았으면 아무것도 하지 않음
    if (!Config.instance.isDebugMode) {
      print("writeDebug is only available in debug mode.");
      return;
    }
    // if (state is! UIStateUsbPortConnected && state.port == null) {
    //   showScafold("Debug Write Failed: Port not connected.");
    //   print("Debug Write Failed: Port not connected.");
    //   return;
    // }
    // 2. 메시지에 '#' 종료 문자가 없으면 추가해줍니다.
    String commandToSend = msg.endsWith('#') ? msg : '$msg#';

    // 3. 문자열을 Uint8List로 변환합니다.
    final dataToSend = Uint8List.fromList(commandToSend.codeUnits);

    // 4. 실제 포트로 데이터를 씁니다.
    write(dataToSend);
  }

  void showScafold(String data) {
    if(Config.instance.isDebugMode) {
      Fluttertoast.showToast(
          msg: data,
          gravity: ToastGravity.CENTER,
          timeInSecForIosWeb: 1,
          backgroundColor: Colors.red,
          textColor: Colors.white,
          fontSize: 16.0
      );
    }
  }

  void _processBuffer() {
    // 버퍼에 종료 문자가 포함되어 있는 동안 계속 반복
    while (_receiveBuffer.toString().contains(_terminator)) {
      final bufferString = _receiveBuffer.toString();
      final messageEndIndex = bufferString.indexOf(_terminator);

      // 4. 종료 문자까지의 완전한 메시지를 추출
      final fullMessage = bufferString.substring(0, messageEndIndex);
```

```
718         print("Complete message parsed: $fullMessage");
719 
720         // 5. 추출된 메시지를 처리 로직으로 전달
721         listenByPort(fullMessage);
722 
723         // 6. 처리된 메시지와 종료 문자를 버퍼에서 제거
724         // substring(messageEndIndex + 1)을 통해 버퍼의 나머지 부분을 유지
725         final remaining = bufferString.substring(messageEndIndex + 1);
726         _receiveBuffer.clear();
727         _receiveBuffer.write(remaining);
728       }
729     }
730 
731     void write(Uint8List dataToSend) {
732       final String dataAsString = String.fromCharCodes(dataToSend);
733       showScafold("Write to port: $dataAsString");
734       print("write to port: $dataAsString");
735       state.port?.write(dataToSend);
736     }
737 
738     //Loading 등이면 초기화 처리 진행
739     void initPortState() {
740       if(state.connectedDevice == null) {
741         state = UIStateUsbPort.init(
742             availablePorts: state.availablePorts,
743             connectedDevice: state.connectedDevice,
744             port: state.port,
745             lastCommand: null,
746         );
747         return;
748       }
749 
750       state = UIStateUsbPort.connected(
751           availablePorts: state.availablePorts,
752           connectedDevice: state.connectedDevice!,
753           port: state.port!,
754           lastCommand: null,
755       );
756     }
757 
758 }
```

1 파일 개요

클래스: `SerialPortVM extends _$SerialPortVM`
역할:

- Flutter + STM32 (USB Serial) 통신 관리
- 비활성 상태 감지(SLEEP) 및 재연결 로직 구현
- 명령 전송/응답 처리, 상태 관리(UIStateUsbPort)
- Riverpod `@Riverpod(keepAlive: true)` → 앱 전체에서 유지

2 주요 변수 및 구조

변수	설명
`_subscription`	USB 포트 스트림 구독
`_receiveBuffer`	수신 데이터를 임시 저장
`_terminator`	메시지 종료 문자 (`#`)
`formatRegex`	O, W, E 포맷 체크 정규식
`_inactivityTimer`	비활성 상태 감지 후 자동 SLEEP 명령
`_reconnectTimer`	연결 끊김 시 재연결 타이머
`_reconnectAttempts`	현재 재연결 시도 횟수
`_maxReconnectAttempts`	최대 재연결 횟수 (10)

3 핵심 기능

3.1 포트 연결 및 초기화

```
void _init() => connectPort();
```

- `_subscription` 취소 및 포트 닫기
- `UsbSerial.listDevices()`로 장치 탐색
- STM32/USB Serial 자동 선택 또는 다중 장치 선택 다이얼로그
- 연결 성공 시 `UIStateUsbPortConnected` 상태로 변경
- **디버그 모드**에서는 가상 포트 시뮬레이션 가능

3.2 수신 처리

- `_receiveBuffer`에 데이터 누적 → `_processBuffer()` 호출
- 종료 문자(`#`) 단위로 완전한 메시지 추출
- `listenByPort(fullMessage)` 호출하여 명령별 처리

명령별 처리 예시:

lastCommand	수신 시 처리
handshake	OK → Step1 화면, FAIL → Splash, FULL → 5초 토스트 후 Splash
cmdPhone	OK → Step2, OPEN → OpeningDoor, FAIL → 에러 토스트, REJECT → 5회 실패 시 Splash
open	OK → Step3

lastCommand	수신 시 처리
close	O, W, E 포맷 → Step4 화면 이동
postper	OK → postData 전송
postData	OK → Splash
recheck	O, W, E 포맷 → Step4 재실행

3.3 명령 전송

- `writeToPort(PORT_COMMANDS command)`
  - 자동 종료 문자가 없으면 `#` 추가
  - 쓰기 시 `_resetInactivityTimer()` 호출 (SLEEP 제외)
- `writeToPortPhone(String phoneCommand)` → 전화번호 전송
- `okay()` / `okayNextStep()` → `postper` → `postData` 재시도 로직 포함

3.4 비활성 상태 감지

```
void _resetInactivityTimer() {
  _inactivityTimer?.cancel();
  _inactivityTimer = Timer.periodic(
    Duration(seconds: 120),
    (timer) {
      showScafold(AppStrings.trySleep);
      writeToPort(PORT_COMMANDS.sleep);
    }
  );
}
```

- 2분 동안 명령이 없으면 자동으로 SLEEP 명령 전송

3.5 재연결 로직

```
void _handleDisconnection() { _attemptReconnect(); }

void _attemptReconnect() {
  if (_reconnectAttempts >= _maxReconnectAttempts) return;
  _reconnectAttempts++;
  _reconnectTimer = Timer(Duration(seconds: 3), () {
    connectPort();
  });
}
```

- 연결 실패 시 자동 재연결

- 최대 10회 재시도
- 성공 시 `_reconnectAttempts` 초기화

3.6 디버그 기능

- `listenDebug(String data)` → 수신 데이터 시뮬레이션
- `writeDebug(String msg)` → 포트에 디버그 명령 전송
- `_showScafold()` → 디버그 모드에서 FlutterToast로 메시지 표시

3.7 상태 초기화

```
void initPortState() {
  if(state.connectedDevice == null) {
    state = UIStateUsbPort.init(...);
  } else {
    state = UIStateUsbPort.connected(...);
  }
}
```

- Loading 상태에서 포트 상태 초기화
- 연결이 유지되어 있는지 확인

4️⃣ 특징 및 장점

1. **자동 재연결 + 최대 재시도** → 안정적인 포트 연결
2. **비활성 상태 감지(SLEEP)** → MCU 절전 모드 자동 전환
3. **명령별 응답 처리** → UI 화면 전환 및 토스트 메시지 연동
4. **디버그 모드** → 실제 장치 없이 개발 가능
5. **Riverpod 상태 관리** → UI와 포트 상태 실시간 동기화

5️⃣ 개선/확장 아이디어

- `_inactivityTimer`와 `_reconnectTimer`를 `Timer.periodic` → `Timer`+`Duration` 조합으로 관리하여 중복 방지
- `_receiveBuffer`의 문자열 처리 최적화 → StringBuffer 대신 Queue 사용 가능
- 장치 선택 UI → StreamBuilder/Provider와 연결하여 실시간 장치 변경 지원
- 명령별 응답 처리 → Map<PORT_COMMANDS, Function> 구조로 리팩토링 가능