Focus & TTS로 구현한 3계층 접근성 패턴

시각 장애인도 사용 가능한 앱 만들기
Barrier-Free Kiosk

이상훈
WALDLUST

목차

본 발표에서는 시각 장애인도 사용할 수 있는 키오스크 앱 개발에 적용한 3계층 접근성 패턴을 소개합니다.
실제 적용 사례와 개발 팁도 함께 공유합니다.

PDF

발표자 소개

- 이상훈, 발트루스트
- Barrier-Free Kiosk 프로젝트
- 접근성 개발 경험

Flutter

- FocusScope
- FocusManager
- Ink + InkWell

Focus 기술 구현

- Focus 관리 방법
- 화면 요소 포커싱 기법
- 사용자 인터랙션 처리

3계층 접근성 패턴

- KeyboardHandler
- AreaFocusGroup
- FocusCustom

배리어프리 키오스크

법적 의무화

- 2025년 1월 28일부터 50제곱미터 이상의 일반 사업장에서 신규 설치하는 키오스크에 적용되며, 위반 시 최대 3천만원의 과태료가 부과
- 장애인차별금지법 개정안의 소상공인 부담을 완화하기 위해, 2025년 8월 복지부가 50㎡ 이하의 소상공인은 배리어프리 키오스크 설치 의무에서 제외하는 시행령 개정안을 입법 예고

적용대상 단계적 범위

- 1단계: (24년 1월 28일 시행) 공공·교육·의료기관, 이동·교통시설 등
- 2단계: (24년 7월 28일 시행) 복지시설, 상시 100인 이상 매장
- 3단계: (25년 1월 28일 시행) 상시 100인 미만 매장 확대 적용

Physical Keyboard

onTap & Enter

- 물리적 Enter 키와 onTap 효과를 동일하게 유지

영역 간 이동(F17/F18 키)

- 상하 이동
- 그룹핑 및 영역 단위 이동

요소 간 이동(F19/F20 키)

- 개별 위젯 별 이동
- 단일 그룹에 이동시 개별 포커스 적용

FocusScope FocusManager

FocusScope

- 로컬(위젯 트리 내부) 포커스 관리
- context 기반 접근 방식 (FocusScope.of(context))
- 특정 UI 컴포넌트 내부로 포커스 제한 가능
- 폼, 다이얼로그 등 격리된 UI 요소에 적합
- 위젯 트리 구조에 의존적
- 부모-자식 관계의 명시적 역할

FocusManager

- 글로벌(앱 전체) 포커스 관리
- 싱글톤 인스턴스 접근 방식 (FocusManager.instance)
- 화면 전환 시에도 일관된 포커스 관리 가능
- 하드웨어 키보드 이벤트 처리와 접근성 기능 연동에 유리
- 중앙 집중식 포커스 정책 구현에 적합

FocusNode
FocusScopeNode

FocusNode

- 개별 위젯(입력필드, 버튼 등)의 포커스 상태 관리
- FocusNode 직접 생성 및 할당, 다양한 커스텀 핸들링 가능
- 포커스 제어 메서드 제공
- requestFocus(),
- unfocus()
- hasFocus

FocusScopeNode

- 여러 FocusNode를 하나의 영역(스코프)로 묶어 포커스 경계 관리
- context 기반 접근 FocusScope.of(context)
- 스코프 내 마지막 포커스 지점 기억 및 자동 복귀 기능 제공

FocusTraversalGroup

위젯 트리에서 부모 FocusNode 아래에 자식 FocusNode가 있으면
Flutter는 "FocusNode의 그룹(서브트리)"로 인식

OrderedTraversalPolicy TraversalOrder

탐색 정책과 그룹 경계를 설정

부모-자식 포커스 트리 생성 X

3계층 포커스 구조

1

KeyboardHandler
전역 포커스 관리자

2

AreaFocusGroup
영역별 이동 관리자

3

FocusCustom
개별 요소 관리자


개발자는 코드로
대화합니다!


그래
가보자고
가자니까
그래 가자고

3계층 포커스 구조 (1)

KeyboardHandler
전역 포커스 관리자

KeyboardHandler는 싱글톤 패턴을 활용하여 앱 전체의 키보드 이벤트를 중앙 집중식으로 처리

HardwareKeyboard.instance.addHandler를 통해 물리 키보드 이벤트를 감지하고, 포커스 이동과 관련된 모든 로직을 관리

- 한 곳에서 모든 포커스 이동 로직 관리
- 화면별 포커스 그룹만 등록/해제
- 하드웨어 키 이벤트 한 곳에서 처리

```dart
@override
void initState() {
  super.initState();
  WidgetsBinding.instance.addObserver( observer: this);

  /// 윈도우 매니저 리스너
  if (Platform.isWindows) {
    windowManager.addListener( listener: this);
    /// WindowsPlatform에 글로벌 컨테이너 설정
    WindowsPlatform.setGlobalContainer( container: ProviderScope.containerOf(context));
  }

  /// 키보드 핸들러 초기화
  WidgetsBinding.instance.addPostFrameCallback( callback: ( Duration _) async {
    try {
      /// 기존 핸들러 제거
      await Future.delayed( duration: const Duration(milliseconds: 500));
      HardwareKeyboard.instance.removeHandler( handler: KeyboardHandler.handleKeyEvent);

      /// 키보드 초기화 및 핸들러 등록
      await KeyboardHandler.initialize( container: ProviderScope.containerOf(context));
      await Future.delayed( duration: const Duration(milliseconds: 100));
      /// 앱 최상단에서 키보드 이벤트 처리
      HardwareKeyboard.instance.addHandler( handler: KeyboardHandler.handleKeyEvent);

      debugPrint('키보드 핸들러 등록 완료');
    } catch (e) {
      debugPrint('키보드 핸들러 초기화 실패: $e');
    }
  });
}
```

```dart
abstract class KeyboardHandler {
  No usages
  KeyboardHandler._();

// 현재 활성화된 포커스 그룹의 key(화면 식별자)들을 스택 구조로 관리
// 최근에 등록된(즉, 현재 화면에 가까운) key가 스택의 마지막에 위치
  10 usages
  static final List<String> _focusKeyStack = [];

// 각 화면(route)별로 등록된 포커스 그룹 설정을 저장하는 맵
// key: routeName(화면 식별자), value: 해당 화면의 FocusGroupConfig 리스트
  12 usages
  static final Map<String, List<FocusGroupConfig>> focusGroupConfigMap = {};

// 현재 활성 포커스 그룹 내에서 몇 번째 그룹(인덱스)에 포커스가 있는지 추적
// 포커스 이동 시 이 인덱스를 갱신하여 그룹 내 포커스 순서를 관리
  16 usages
  static int _currentGroupIndex = 0;
```

```dart
/// 현재 활성 포커스 그룹 반환
static List<FocusGroupConfig>? get _currentFocusGroupConfigs {
  if (_focusKeyStack.isEmpty) return null;
  final activeKey = _focusKeyStack.last;
  return focusGroupConfigMap[activeKey];
}
```

```dart
1 usage
static void _performInitialRegistration(
  String key,
  List<FocusGroupConfig> focusConfigs,
) {
  focusGroupConfigMap[key] = focusConfigs;

  // 메인 스크린은 항상 0번 인덱스에 고정
  if (key == MainScreen.routeName) {
    _focusKeyStack.insert( index: 0,  element: key);
  } else {
    _focusKeyStack.add( value: key);
  }

  _currentGroupIndex = 0;

  debugPrint('포커스 그룹 등록 -> Key: $key'
      ' 총 그룹 수: ${focusConfigs.length}'
      ' 스택 위치: ${_focusKeyStack.indexOf( element: key)}'
      ' 총 스택 크기: ${_focusKeyStack.length}');
}
```

```dart
static bool handleKeyEvent(KeyEvent event) {
    return true;
  /// true의 경우 기본 동작을 거부하고 직접 처리
    case PhysicalKeyboardKey.f24:
      /// 뒤로가기
      _onLeftButton();
      return true;
    case PhysicalKeyboardKey.enter:
      _onEnterButton();
      return false;
    case PhysicalKeyboardKey.f17:
      _moveToGroup(isUpward: true);
      return true;
    case PhysicalKeyboardKey.f18:
      _moveToGroup(isUpward: false);
      return true;
    case PhysicalKeyboardKey.f19:
      handleTabFocus(isRight: false);
      return true;
    case PhysicalKeyboardKey.f20:
      handleTabFocus(isRight: true);
      return true;
    case PhysicalKeyboardKey.audioVolumeUp:
      _onVolumeUpButton(); // 커스텀 메서드 호출
      return true;
    case PhysicalKeyboardKey.audioVolumeDown:
      _onVolumeDownButton(); // 커스텀 메서드 호출
      return true;
    default:
      return false;
  }
}
```

```dart
2 usages
static void _moveToGroup({required bool isUpward}) {
  final List<FocusGroupConfig>? configs = _currentFocusGroupConfigs;
  if (configs == null || configs.isEmpty) return;

  final int lastIndex = configs.length - 1;

  /// 현재 포커스된 노드와 그룹 인덱스 동기화
  if (isUpward) {
    _currentGroupIndex = (_currentGroupIndex > 0)
        ? _currentGroupIndex - 1
        : lastIndex;
  } else {
    _currentGroupIndex = (_currentGroupIndex + 1) % configs.length;
  }

  debugPrint('그룹 이동: 현재 인덱스 $_currentGroupIndex / $lastIndex');

  // 그룹 대표 노드로 포커스 이동
  final FocusGroupConfig currentConfig = configs[_currentGroupIndex];
  currentConfig.focusNode.requestFocus();

  // 단일 자식 그룹이면 자동 포커스 이동
  WidgetsBinding.instance.addPostFrameCallback( callback: ( Duration  _) {
    if (currentConfig.isSingleChild) {
      _moveToFirstFocusableInGroup();
      debugPrint('✅ 단일 자식 그룹: 자동 포커스 이동');
    }
  });
}
```

```dart
/// 그룹 내 첫 번째 포커스 가능한 요소로 이동
static void _moveToFirstFocusableInGroup() {
  final currentFocus = FocusManager.instance.primaryFocus;
  if (currentFocus != null) {
    // 현재 그룹 내에서 다음 포커스 가능한 요소 찾기
    final scope = currentFocus.enclosingScope;
    scope?.nextFocus();
  }
}
```

```dart
@override
Widget build(BuildContext context) {
  final useMultiLanguage = ref.watch(settingsProvider).useMultiLanguage;

  return Scaffold(
    body: FocusTraversalGroup(
      policy: OrderedTraversalPolicy(),
      child: Column(
        children: [
          LowPostureHeight(),
          /// 앱바 영역 - 그룹 대표 노드로 감싸기
          AreaFocusGroup(
            order: 1,
            focusNode: _appBarGroupFocus,
            customTtsMessage: useMultiLanguage ? "상단 메뉴 영역입니다. 언어 변경, 홈 버튼 등이 있습니다."
                : "상단 메뉴 영역입니다. 음량 조절, 홈 버튼 등이 있습니다.",
            child: HomeBarrierAppBar(),
          ), // AreaFocusGroup

          /// 카테고리 영역
          AreaFocusGroup(
            order: 2,
            focusNode: _categoryGroupFocus,
            customTtsMessage: "상품 카테고리 영역입니다. 좌우 방향키로 카테고리를 선택할 수 있습니다.",
            child: HomeCategoryBarrier(controller: _controller),
          ), // AreaFocusGroup

          /// 상품 리스트
          AreaFocusGroup(
            order: 3,
            focusNode: _productListGroupFocus,
            customTtsMessage: "상품 목록 영역입니다. 좌우 방향키로 상품을 둘러보세요.",
            child: HomeProductListBarrier(controller: _controller),
          ), // AreaFocusGroup
```

```dart
class _HomeBarrierScreenState extends ConsumerState<HomeBarrierScreen> {
  late final ScrollController _controller;

  // 각 영역별 그룹 대표 FocusNode
  final FocusNode _appBarGroupFocus = FocusNode(); // 1. 앱바
  final FocusNode _categoryGroupFocus = FocusNode(); // 2. 카테고리
  final FocusNode _productListGroupFocus = FocusNode(); // 3. 상품 리스트
  final FocusNode _cartGroupFocus = FocusNode(); // 4. 카트
  final FocusNode _barrierGroupFocus = FocusNode(); // 5. BarrierFreeList

  @override
  void initState() {
    super.initState();
    _controller = ScrollController();

// FocusGroupConfig로 포커스 그룹 등록
    final focusConfigs = <FocusGroupConfig>[
      FocusGroupConfig(
        focusNode: _appBarGroupFocus,
        isSingleChild: false,
      ), // FocusGroupConfig
      FocusGroupConfig(
        focusNode: _categoryGroupFocus,
        isSingleChild: false,
      ), // FocusGroupConfig
      FocusGroupConfig(
        focusNode: _productListGroupFocus,
        isSingleChild: false,
      ), // FocusGroupConfig
      FocusGroupConfig(
        focusNode: _cartGroupFocus,
        isSingleChild: false,
      ), // FocusGroupConfig
      FocusGroupConfig(
        focusNode: _barrierGroupFocus,
        isSingleChild: false,
      ), // FocusGroupConfig
    ]; // <FocusGroupConfig>[]

    // 포커스 그룹 등록
    KeyboardHandler.registerFocusGroups(HomeBarrierScreen.routeName, focusConfigs);
```

```dart
@override
void dispose() {
  _controller.dispose();

  KeyboardHandler.unregisterFocusGroups( key: HomeBarrierScreen.routeName);
  _appBarGroupFocus.dispose();
  _categoryGroupFocus.dispose();
  _productListGroupFocus.dispose();
  _cartGroupFocus.dispose();
  _barrierGroupFocus.dispose();

  super.dispose();
}
```

3계층 포커스 구조 (2)

AreaFocusGroup
영역별 이동 관리자

화면을 논리적 영역으로 분할하여 각 영역별 독립적 포커스 관리와 시각적 피드백 제공

각 영역별 고유한 customTtsMessage 설정

영역 포커스(_hasAreaFocus)와 자식 포커스(_hasChildFocus) 별도 추적

```dart
class _AreaFocusGroupState extends ConsumerState<AreaFocusGroup> {
  6 usages
  bool _hasAreaFocus = false;
  6 usages
  bool _hasChildFocus = false;

  60 usages
  @override
  void initState() {
    super.initState();
    widget.focusNode.addListener( listener: _onAreaFocusChange);
    // FocusManager 리스너 추가 - 자식 포커스 감지용
    FocusManager.instance.addListener( listener: _onGlobalFocusChange);
  }
```

```dart
4 usages
void _onAreaFocusChange() {
  final bool currentAreaFocus = widget.focusNode.hasFocus;
  if (_hasAreaFocus != currentAreaFocus) {
    setState(() => _hasAreaFocus = currentAreaFocus);

    // 영역 포커스를 받을 때만 TTS 재생
    if (currentAreaFocus && !_hasChildFocus && !widget.isSingleChild) {
      _speakAreaMessage();
    }
  }
}

2 usages
void _onGlobalFocusChange() {
  final FocusNode? currentFocus = FocusManager.instance.primaryFocus;
  if (currentFocus == null) return;

  // 현재 포커스가 이 영역의 자식인지 확인
  bool isChildFocus = _isDescendantOf(currentFocus, areaNode: widget.focusNode) &&
      currentFocus != widget.focusNode;

  if (_hasChildFocus != isChildFocus) {
    setState(() => _hasChildFocus = isChildFocus);

    // 자식에서 영역으로 돌아올 때 TTS 재생
    if (!isChildFocus && _hasAreaFocus) {
      _speakAreaMessage();
    }
  }
}
```

```dart
  No usages
  @override
  Widget build(BuildContext context) {
    return FocusTraversalOrder(
      order: NumericFocusOrder( order: widget.order),
      child: FocusTraversalGroup(
        child: Focus(
          focusNode: widget.focusNode,
          skipTraversal: true,
          child: Stack(
            children: [
              // 실제 위젯 내용
              widget.child,
              // 영역 포커스 시에만 오버레이 표시 (자식 포커스 시 숨김)
              if (_hasAreaFocus && !_hasChildFocus) _buildAreaFocusOverlay(),
            ],
          ), Stack
        ), Focus
      ), FocusTraversalGroup
    ); FocusTraversalOrder
  }
```

```dart
/// 영역 포커스 오버레이 생성
1 usage
Widget _buildAreaFocusOverlay() {
  return Positioned.fill(
    child: IgnorePointer(
      child: DecoratedBox(
        decoration: BoxDecoration(
          border: Border.all(
            color: widget.focusBorderColor,
            width: widget.focusBorderWidth,
          ), Border.all
          borderRadius: BorderRadius.circular( radius: widget.focusBorderRadius),
        ), BoxDecoration
      ), DecoratedBox
    ), IgnorePointer
  ); Positioned.fill
}
```

3계층 포커스 구조 (3)

FocusCustom
개별 요소 관리자

FocusCustom - 개별 UI 요소를 포커스 가능한 위젯으로 변환하여 키보드 네비게이션과 TTS 안내를 통합 제공

InkWell과 Material을 활용한 터치 피드백과 포커스 상태별 시각적 오버레이(파란색 테두리) 제공

Enter 키 이벤트 처리로 키보드만으로 버튼 클릭 동작 구현, 사운드 효과와 함께 일관된 UX 제공

FocusNode를 통해 개별 요소의 포커스 상태 관리 및 onFocusChange 콜백으로 외부 상태 연동

```dart
class FocusCustom extends ConsumerStatefulWidget {
  final Widget child;
  final String text;
  final VoidCallback? onTap;
  final bool isFilterTts;
  final bool? autoFocus;
  final double inkBorderRadius;
  final double? height;
  final Decoration decoration;

  final Function(bool)? onFocusChange;

  const FocusCustom({
    super.key,
    required this.child,
    required this.text,
    this.decoration = const BoxDecoration(),
    this.onTap,
    this.isFilterTts = false,
    this.autoFocus = false,
    this.inkBorderRadius = 16.0,
    this.height,
    this.onFocusChange,
  });

  @override
  ConsumerState<FocusCustom> createState() => _FocusCustomState();
}

class _FocusCustomState extends ConsumerState<FocusCustom> {
  late FocusNode _focusNode;

  static final TouchSoundService _soundService = TouchSoundService();

  @override
  void initState() {
    super.initState();
    _focusNode = FocusNode();
    _focusNode.addListener(_onFocusChange);
    _handleAutoFocus();
  }
```

```dart
  void _handleAutoFocus() {
    if (widget.autoFocus == true) {
      WidgetsBinding.instance.addPostFrameCallback((_) {
        if (mounted && ref.read(voiceGuideProvider).voiceGuideEnabled) {
          _focusNode.requestFocus();
        }
      });
    }
  }

  void _onFocusChange() {
    if (_focusNode.hasFocus) {
      _speakMessage();
      // 포커스 획득 시 콜백 실행
      if (widget.onFocusChange != null) {
        widget.onFocusChange!(true);
      }
    } else {
      // 포커스 해제 시 콜백 실행
      if (widget.onFocusChange != null) {
        widget.onFocusChange!(false);
      }
    }
  }

  void _speakMessage() {
    final voiceGuideEnabled = ref.read(voiceGuideProvider).voiceGuideEnabled;
    if (voiceGuideEnabled) {
      TTSUtil().forceSpeak(widget.text, isFilter: widget.isFilterTts);
    }
  }

  void _handleTap() {
    if (widget.onTap != null) {
      _soundService.playClick();
      widget.onTap!();
    }
  }
```

```dart
  @override
  Widget build(BuildContext context) {
    return Focus(
      focusNode: _focusNode,
      onKeyEvent: _handleKeyEvent,
      child: ExcludeFocus(
        child: SizedBox(
          height: widget.height,
          child: Stack(
            children: [
              Material(
                color: Colors.transparent,
                child: Ink(
                  decoration: widget.decoration,
                  child: InkWell(
                    borderRadius: BorderRadius.circular(widget.inkBorderRadius),
                    onTap: _handleTap,
                    child: widget.child,
                  ), // InkWell
                ), // Ink
              ), // Material
              if (_focusNode.hasFocus) _buildFocusOverlay(),
            ],
          ), // Stack
        ), // SizedBox
      ), // ExcludeFocus
    ); // Focus
  }

  Widget _buildFocusOverlay() {
    return Positioned.fill(
      child: DecoratedBox(
        decoration: BoxDecoration(
          border: Border.all(color: Colors.blue, width: 2),
          borderRadius: BorderRadius.circular(8),
        ), // BoxDecoration
      ), // DecoratedBox
    ); // Positioned.fill
  }
```

```dart
  KeyEventResult _handleKeyEvent(FocusNode node, KeyEvent event) {
    if (event is KeyDownEvent && _focusNode.hasFocus) {
      if (event.logicalKey == LogicalKeyboardKey.enter) {
        _handleTap();
        return KeyEventResult.handled;
      }
    }
    return KeyEventResult.ignored;
  }
```

```dart
/// Tab/Shift+Tab 기능 - 포커스 이동 (isRight: true면 다음, false면 이전)
2 usages
static void handleTabFocus({required bool isRight}) {
  final FocusNode? currentFocus = FocusManager.instance.primaryFocus;
  if (currentFocus == null) {
    print(object: '현재 포커스된 위젯이 없습니다');
    return;
  }
  if (isRight) {
    currentFocus.nextFocus();
  } else {
    currentFocus.previousFocus();
  }
}
```

```dart
1 usage
static void _syncGroupIndexWithCurrentFocus() {
  final FocusNode? currentFocus = FocusManager.instance.primaryFocus;
  if (currentFocus == null || _currentFocusGroups == null) return;

  // 현재 포커스된 노드가 어떤 그룹에 속하는지 찾기
  for (int i = 0; i < _currentFocusGroups!.length; i++) {
    final FocusNode groupNode = _currentFocusGroups![i];

    // 현재 포커스가 그룹 대표 노드이거나, 그룹 대표 노드의 하위 요소인지 확인
    if (currentFocus == groupNode ||
        _isDescendantOf(currentFocus, groupNode)) {
      if (_currentGroupIndex != i) {
        final int oldIndex = _currentGroupIndex;
        _currentGroupIndex = i;
        debugPrint('🔄 그룹 인덱스 동기화: $oldIndex → $_currentGroupIndex');
      }
      return;
    }
  }
}
```

Ink + InkWell

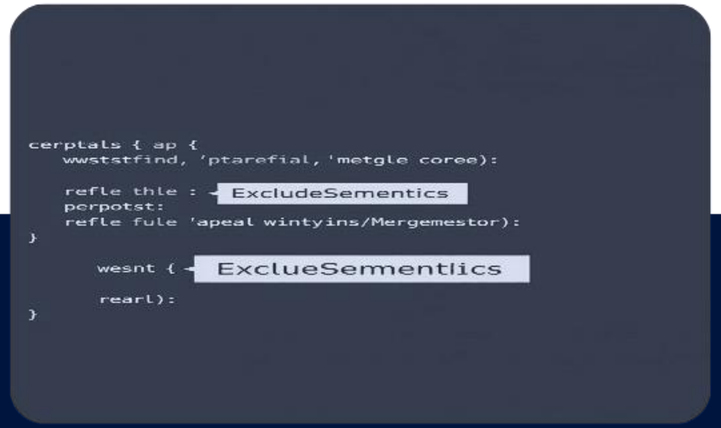
위젯 트리가 렌더 오브젝트 트리로 변환되고,
렌더 오브젝트는 자신만의 페인팅 순서와 레이어를 가지고 있어, 자식이라도 Container가
InkWell의 잉크 효과를 덮어버려서 보이지 않게 만듦

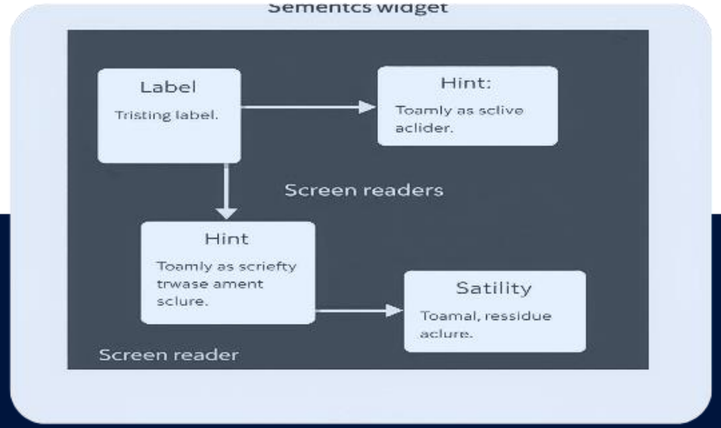
개발 과정의 트러블 슈팅

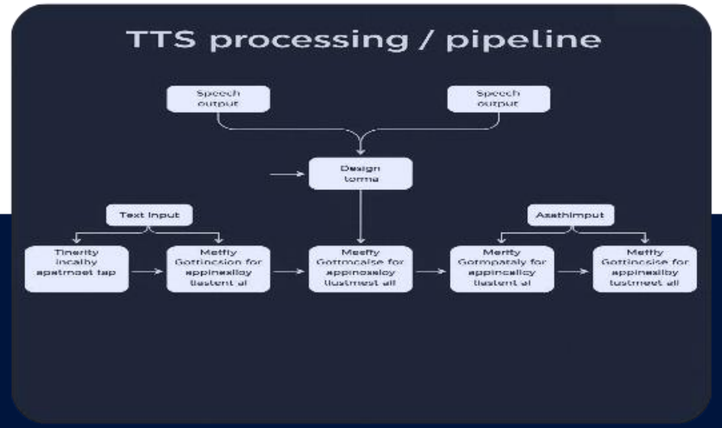
포커스 이동 문제 해결과 접근성 네비게이션 개선

1	Arrow 키 이벤트 직접 처리	- Focus 노드가 있어야만 포커스 처리 되는 것을 확인 - 포커스 이동 순서가 논리적으로 보장 x - 상하 이동 시, 일관성 x - 포커스를 받아도 확인 불가능한 경우 존재
2	FocusCutom	- 상하 이동 시, 일관성 x - 영역 단위 이동 불가능 - 영역 별 테두리 표시 불가능
3	AreaFocusGroup	- 단일 위젯 영역 처리 불가능 - 개별 - 영역 간 이동 동기화 불가능
4	KeyboardHandler	- 요구사항 총족

Flutter 접근성 개발 팁







배리어프리 개발 패키지

- volume_controller
- flutter_tts
- flutter_soloud

ExcludeFocus

- excluding : false
- true면 하위 포커스 제외

돋보기(Magnifier)

- RawMagnifier 위젯 사용

Q&A

접근성 패턴 구현 시 가장 어려웠던 점은 무엇인가요?

pub.dev에 패키지화 후 올릴 계획이 있나요?

waldlust 키오스크

- 관련문의 : contact@waldlust.co.kr
- 홈페이지 : https://waldsolution.com

접근성 관련 리소스

- Flutter 공식 접근성 가이드
- WCAG (웹 콘텐츠 접근성 지침)
- 애플 및 구글의 모바일 접근성 가이드라인
- A11Y Project (웹 접근성 커뮤니티)

연락처 및 커뮤니티

- 이메일: oasa741@gmail.com
- 깃허브: @FlutterNeverDie
- 블로그: https://hooninha.tistory.com/

감사합니다