

JI

Jordi Ibor Quintana

Pro plan

[ld_wbench5/](#)

Updating ld_wbench5 Flutter Project

Share

JI

Tens al projecte el repositori de codi del projecte ld_wbench5 en que estem treballant plegats.

També tens l'última conversa que hem mantingut fins que el codi ha quedat a mitges.

Sisplau, posa't al dia en tot el codi del projecte i llegeix la conversa per a entendre on som i què estem fent.

Edit

I've reviewed the project code and our previous conversation. Let me summarize where we are and what we're working on:

Current Project Status

This is a Flutter application called "Sabina" with a custom MVC architecture. We've been implementing a visual theming system and have discovered a fundamental issue with the architecture.

Key Components:

1. **Widgets:** Custom widgets like `LdButton`, `LdTextField`, `LdThemeSelector`, etc.
2. **Services:** Centralized services like `LdTheme`, `ThemeService`, `L` (for localization)
3. **Core Architecture:** Custom MVC pattern with `LdWidgetAbs`, `LdWidgetCtrlAbs`, and `LdModelAbs`

Recently Completed:

- Implemented a comprehensive theming system with `LdTheme` service
- Created UI components for theme selection and visualization
- Implemented internationalization with the `L` service

Current Issue:

There's a fundamental misunderstanding in the architecture about Flutter's lifecycle. The issue is:

1. In the current architecture, you're creating models and controllers in the widget constructor and storing them as properties of the `StatefulWidget`.
2. This conflicts with Flutter's lifecycle where `StatefulWidget` is immutable and can be recreated frequently, while `State` is persistent.
3. This leads to an error where the same `State` instance is being reused across different widget instances.

Solution We're Working On:

We're refactoring the architecture to:

1. Make the `StatefulWidget` lightweight, only containing configuration data (possibly in a map)
2. Move state management to the `State` class (your controllers)
3. Have controllers create and manage models
4. Use a map-based approach for configuration with prefixes:
 - `mf` : Model Fields (data)
 - `cf` : Controller Fields (behavior)
 - `ef` : Event Fields (events)

We were in the process of implementing a `MapManager` to optimize memory use by sharing maps between instances.

Next Steps

Based on our conversation, we need to:

1. Complete the refactoring of the architecture to fix the lifecycle issue
2. Implement the map-based configuration approach with proper field prefixing
3. Create a central map manager to optimize memory usage
4. Update existing widgets to follow this new pattern

Would you like me to focus on any specific part of this implementation next?

Retry

JI

Primer, com en cada conversa que tenim, t'he de demanar que em responguis en la mateixa llengua en que em dirigeixo a tú.

Edit

Ho entenc perfectament. A partir d'ara et respondré en català, que és la llengua en què t'estàs dirigint a mi. Així mantindrem una conversa més fluida i natural.

He revisat tot el codi del projecte i la nostra conversa anterior. Estic al dia amb l'estat actual del projecte Sabina i la refactorització de l'arquitectura que estem duent a terme.

Digues-me en què vols que em centri ara, i continuarem treballant en la mateixa línia per resoldre els problemes d'arquitectura que hem identificat.

Retry

JI

Com la conversa va quedar tallada pel teu límit just quan estàvem fent la reconversió del codi base dels widgets i pàgines la versió dels fitxers de codi al repositori (que és la última) no és coherent i té errors.

El primer és que trobis el servei que vas escriure per a gestionar les instàncies de mapes per a alleugerir les classes i evitar duplicitats. La classe finalment es va anomenar 'MapsService'.

Edit

Després de revisar tots els fitxers del repositori, no trobo cap classe anomenada `MapsService` directament. El que sí que trobo és una classe similar anomenada `ModelService` al fitxer `lib/services/model_service.dart` que sembla que és la que estàvem desenvolupant per gestionar les instàncies de mapes.

Aquí tens el contingut del fitxer `lib/services/model_service.dart` que sembla ser el servei que mencionaves:

dart

```
// lib/core/services/model_service.dart
// Servei centralitzat d'instàncies de models de dades.
// Created: 2025/05/09 dv. CLA[JIQ]
```

```
import 'package:ld_wbench5/ui/extensions/map_extensions.dart';
import 'package:ld_wbench5/utils/debug.dart';
```

```
/// Servei centralitzat d'instàncies de models de dades.
```

```
class ModelService {
  /// Instància singleton
  static final ModelService _inst = ModelService._();
  static ModelService get s => _inst;
```

```
  // Membres de la instància -----
  /// Map de mapes indexat per ID
  final LdMap<LdMap<dynamic>> _maps = {};
```

```

    /// Comptador de referències per cada mapa
    final LdMap<int> _refCounts = {};

    /// Informació addicional per cada mapa
    final LdMap<String> _mapTypes = {};

    // Constructors -----
    /// Constructor privat
    ModelService._() {
        Debug.info("MapManager: Inicialitzant gestor de mapes");
    }

    // Registre, Consulta i Liberació ---
    /// Registra un nou mapa i retorna el seu ID
    /// Si ja existeix un mapa idèntic, retorna l'ID d'aquest
    /// [pMap] - El mapa a registrar
    /// [pType] - Tipus del mapa (e.g., "widget", "entity", "config")
    /// [pIdent] - Identificador opcional per ajudar a generar l'ID
    String registerMap(
        LdMap<dynamic> pMap, {
        String pType = "generic",
        String? pIdent,
    }) {
        // Generar un ID per al mapa
        final String mapId = pIdent != null
            ? "${pType}_${pIdent}_${DateTime.now().millisecondsSinceEpoch}"
            : "${pType}_${DateTime.now().millisecondsSinceEpoch}_${_maps.length}";

        // Comprovar si ja existeix un mapa idèntic del mateix tipus
        for (var entry in _maps.entries) {
            if (_mapTypes[entry.key] == pType && _areMapEqual(entry.value, pMap)) {
                // Incrementar el comptador de referències
                _refCounts[entry.key] = (_refCounts[entry.key] ?? 0) + 1;
                Debug.info("MapManager: Reutilitzant mapa existent (ID: ${entry.key},
                    Tipus: $pType, Refs: ${_refCounts[entry.key]})");
                return entry.key;
            }
        }

        // Si no existeix, guardar el mapa
        _maps[mapId] = Map<String, dynamic>.from(pMap);
        _refCounts[mapId] = 1;
        _mapTypes[mapId] = pType;

        Debug.info("MapManager: Nou mapa registrat (ID: $mapId, Tipus: $pType)");
        return mapId;
    }

    /// Obté el mapa per un ID
    LdMap<dynamic> getMap(String pMapId) {

```

```

    assert(!_maps.containsKey(pMapId), "Mapa no trobat: $pMapId");
    return _maps[pMapId]!;
}

/// Allibera una referència a un mapa
/// Si no hi ha més referències, elimina el mapa
void releaseMap(String mapId) {
    if (!_maps.containsKey(mapId)) {
        Debug.warn("MapManager: Intent d'alliberar un mapa inexistent: $mapId");
        return;
    }

    // Decrementar el comptador de referències
    _refCounts[mapId] = (_refCounts[mapId] ?? 1) - 1;

    // Si no hi ha més referències, eliminar el mapa
    if (_refCounts[mapId]! <= 0) {
        _maps.remove(mapId);
        _refCounts.remove(mapId);
        _mapTypes.remove(mapId);
        Debug.info("MapManager: Mapa eliminat (ID: $mapId)");
    } else {
        Debug.info("MapManager: Mapa alliberat (ID: $mapId, Refs restants: $_refCounts[mapId])");
    }
}

/// Funcionalitat interna -----
/// Comprova si dos mapes són iguals (recursivament)
bool _areMapEqual(Map<String, dynamic> pMap1, Map<String, dynamic> pMap2) {
    // ... [mètode complet que pots veure al codi original]
}

/// Comprova si dues llistes són iguals (recursivament)
bool _areListEqual(List pList1, List pList2) {
    // ... [mètode complet que pots veure al codi original]
}

/// Altra funcionalitat -----
/// Obté estadístiques sobre l'ús de mapes
LdMap<dynamic> getStats() {
    // ... [mètode complet que pots veure al codi original]
}

/// Neteja mapes que ja no tenen referències
int cleanup() {
    // ... [mètode complet que pots veure al codi original]
}

/// Crea una còpia d'un mapa existent
String duplicateMap(String pMapId, {String? pNewIdent}) {

```

```
// ... [mètode complet que pots veure al codi original]
}

/// Modifica un mapa existent
/// Retorna el nou ID si es crea un nou mapa (perquè hi ha més referències)
String updateMap(String pMapId, LdMap<dynamic> pUpdates) {
  // ... [mètode complet que pots veure al codi original]
}
}
```

A més a més, en la nostra conversa anterior també estàvem desenvolupant una classe anomenada `WidgetConfigManager` o `MapManager` que sembla que va ser un predecessor o una versió alternativa d'aquest `ModelService`. En la implementació final potser volíem refactoritzar i agrupar aquestes funcionalitats en una sola classe.

Vols que recuperi la versió completa d'aquesta classe tal com la vam estar desenvolupant en la conversa anterior, o vols que continuem amb el `ModelService` que ja està implementat?

Retry

JI

Sí, és aquest. Finalment s'anomena 'MapsService' que és més precís. La resta no canvia en res. Només he canviat els noms de la classe afinant cada vegada més el significat funcional que m'és més lògic. Pots continuar amb el codi de `ModelService` però tenint en compte que finalment s'anomenarà 'MapsService'.

Edit

Entenc que el nom final serà `MapsService`. Perfecte, continuaré amb el codi però tenint en compte aquest canvi de nom.

Reprenem el desenvolupament de l'arquitectura base on ho vam deixar. Estàvem refactoritzant les classes base per adaptar-les al nou paradigma on:

1. Els `StatefulWidget` són lleugers i només tenen configuració
2. Els controladors (que són els `State`) gestionen el cicle de vida i creen els models
3. Utilitzem un gestor centralitzat de mapes (`MapsService`) per optimitzar la memòria

Aquí tens la implementació de `MapsService` (abans `ModelService`):

dart

```
// lib/services/maps_service.dart
// Servei centralitzat de gestió de mapes de dades.
// Created: 2025/05/09 dv. CLA[JIQ]
// Updated: 2025/05/10 ds. CLA - Canvi de nom a MapsService per precisió
```

```

import 'package:ld_wbench5/ui/extensions/map_extensions.dart';
import 'package:ld_wbench5/utils/debug.dart';

/// Servei centralitzat de gestió de mapes de dades.
class MapsService {
  /// Instància singleton
  static final MapsService _inst = MapsService._();
  static MapsService get s => _inst;

  // Membres de la instància -----
  /// Map de mapes indexat per ID
  final LdMap<LdMap<dynamic>> _maps = {};

  /// Comptador de referències per cada mapa
  final LdMap<int> _refCounts = {};

  /// Informació addicional per cada mapa
  final LdMap<String> _mapTypes = {};

  // Constructors -----
  /// Constructor privat
  MapsService._() {
    Debug.info("MapsService: Inicialitzant gestor de mapes");
  }

  // Registre, Consulta i Liberació ---
  /// Registra un nou mapa i retorna el seu ID
  /// Si ja existeix un mapa idèntic, retorna l'ID d'aquest
  /// [pMap] - El mapa a registrar
  /// [pType] - Tipus del mapa (e.g., "widget", "entity", "config")
  /// [pIdent] - Identificador opcional per ajudar a generar l'ID
  String registerMap(
    LdMap<dynamic> pMap, {
    String pType = "generic",
    String? pIdent,
  }) {
    // Generar un ID per al mapa
    final String mapId = pIdent != null
      ? "${pType}_${pIdent}_${DateTime.now().millisecondsSinceEpoch}"
      : "${pType}_${DateTime.now().millisecondsSinceEpoch}_${_maps.length}";

    // Comprovar si ja existeix un mapa idèntic del mateix tipus
    for (var entry in _maps.entries) {
      if (_mapTypes[entry.key] == pType && _areMapEqual(entry.value, pMap)) {
        // Incrementar el comptador de referències
        _refCounts[entry.key] = (_refCounts[entry.key] ?? 0) + 1;
        Debug.info("MapsService: Reutilitzant mapa existent (ID: ${entry.key},
Tipus: $pType, Refs: ${_refCounts[entry.key]})");
        return entry.key;
      }
    }
  }
}

```

```

    }
}

// Si no existeix, guardar el mapa
_maps[mapId] = Map<String, dynamic>.from(pMap);
_refCounts[mapId] = 1;
_mapTypes[mapId] = pType;

Debug.info("MapsService: Nou mapa registrat (ID: $mapId, Tipus: $pType)");
return mapId;
}

/// Obté el mapa per un ID
LdMap<dynamic> getMap(String pMapId) {
    assert(_maps.containsKey(pMapId), "Mapa no trobat: $pMapId");
    return _maps[pMapId]!;
}

/// Allibera una referència a un mapa
/// Si no hi ha més referències, elimina el mapa
void releaseMap(String mapId) {
    if (!_maps.containsKey(mapId)) {
        Debug.warn("MapsService: Intent d'alliberar un mapa inexistent: $mapId");
        return;
    }

    // Decrementar el comptador de referències
    _refCounts[mapId] = (_refCounts[mapId] ?? 1) - 1;

    // Si no hi ha més referències, eliminar el mapa
    if (_refCounts[mapId]! <= 0) {
        _maps.remove(mapId);
        _refCounts.remove(mapId);
        _mapTypes.remove(mapId);
        Debug.info("MapsService: Mapa eliminat (ID: $mapId)");
    } else {
        Debug.info("MapsService: Mapa alliberat (ID: $mapId, Refs restants: $
{_refCounts[mapId]}");
    }
}

// Funcionalitat interna -----
/// Comprova si dos mapes són iguals (recursivament)
bool _areMapEqual(Map<String, dynamic> pMap1, Map<String, dynamic> pMap2) {
    if (pMap1.length != pMap2.length) return false;

    for (final key in pMap1.keys) {
        if (!pMap2.containsKey(key)) return false;

        final value1 = pMap1[key];

```



```

        final value2 = pMap2[key];

        // Ignorar callbacks (no es poden comparar)
        if (value1 is Function && value2 is Function) {
            continue;
        }

        if (value1 is Map && value2 is Map) {
            if (!_areMapEqual(value1 as Map<String, dynamic>, value2 as Map<String,
dynamic>)) {
                return false;
            }
        } else if (value1 is List && value2 is List) {
            if (!_areListEqual(value1, value2)) {
                return false;
            }
        } else if (value1 != value2) {
            return false;
        }
    }

    return true;
}

/// Comprova si dues llistes són iguals (recursivament)
bool _areListEqual(List pList1, List pList2) {
    if (pList1.length != pList2.length) return false;

    for (int i = 0; i < pList1.length; i++) {
        final value1 = pList1[i];
        final value2 = pList2[i];

        if (value1 is Map && value2 is Map) {
            if (!_areMapEqual(value1 as Map<String, dynamic>, value2 as Map<String,
dynamic>)) {
                return false;
            }
        } else if (value1 is List && value2 is List) {
            if (!_areListEqual(value1, value2)) {
                return false;
            }
        } else if (value1 != value2) {
            return false;
        }
    }

    return true;
}

// Altra funcionalitat -----
/// Obté estadístiques sobre l'ús de mapes

```

```

LdMap<dynamic> getStats() {
    final stats = <String, dynamic>{
        'totalMaps': _maps.length,
        'totalReferences': _refCounts.values.fold<int>(0, (sum, count) => sum +
count),
        'mapsByType': <String, int>{},
    };

    // Comptar mapes per tipus
    for (var mapId in _maps.keys) {
        final type = _mapTypes[mapId] ?? "unknown";
        stats['mapsByType'][type] = (stats['mapsByType'][type] ?? 0) + 1;
    }

    return stats;
}

/// Neteja mapes que ja no tenen referències
int cleanup() {
    final keysToRemove = <String>[];

    // Trobar claus per eliminar
    for (var entry in _refCounts.entries) {
        if (entry.value <= 0) {
            keysToRemove.add(entry.key);
        }
    }

    // Eliminar els mapes
    for (var key in keysToRemove) {
        _maps.remove(key);
        _refCounts.remove(key);
        _mapTypes.remove(key);
    }

    Debug.info("MapsService: Neteja completada. ${keysToRemove.length} mapes
eliminats.");
    return keysToRemove.length;
}

/// Crea una còpia d'un mapa existent
String duplicateMap(String pMapId, {String? pNewIdent}) {
    assert(_maps.containsKey(pMapId), "Mapa no trobat: $pMapId");

    final originalMap = _maps[pMapId]!;
    final type = _mapTypes[pMapId] ?? "generic";

    // Registrar una còpia del mapa
    return registerMap(
        Map<String, dynamic>.from(originalMap),

```

```

        pType: type,
        pIdent: pNewIdent,
    );
}

/// Modifica un mapa existent
/// Retorna el nou ID si es crea un nou mapa (perquè hi ha més referències)
String updateMap(String pMapId, LdMap<dynamic> pUpdates) {
    assert(_maps.containsKey(pMapId), "Mapa no trobat: $pMapId");

    // Si hi ha més d'una referència, hem de crear un nou mapa
    if (_refCounts[pMapId]! > 1) {
        // Reduir la referència al mapa original
        _refCounts[pMapId] = _refCounts[pMapId]! - 1;

        // Crear una còpia i aplicar les actualitzacions
        final originalMap = Map<String, dynamic>.from(_maps[pMapId]!);
        final type = _mapTypes[pMapId] ?? "generic";

        // Aplicar les actualitzacions
        originalMap.addAll(pUpdates);

        // Registrar el nou mapa
        return registerMap(
            originalMap,
            pType: type,
        );
    } else {
        // Si només hi ha una referència, podem modificar el mapa directament
        _maps[pMapId]!.addAll(pUpdates);
        return pMapId;
    }
}
}

```

Ara, continuem amb la refactorització de les classes base, començant amb `LdWidgetAbs`. A la nostra conversa anterior, estàvem adaptant-la per utilitzar el gestor de mapes:

dart

```

// lib/core/ld_widget/ld_widget_abs.dart
// Widget base simplificat per a l'aplicació
// Created: 2025/04/29 dt. CLA[JIQ]
// Updated: 2025/05/10 ds. CLA - Adaptació per utilitzar MapsService

import 'package:flutter/material.dart';
import 'package:ld_wbench5/core/ld_model_abs.dart';
import 'package:ld_wbench5/core/ld_taggable_mixin.dart';
import 'package:ld_wbench5/core/map_fields.dart';
import 'package:ld_wbench5/services/maps_service.dart';

```

```

import 'package:ld_wbench5/ui/extensions/map_extensions.dart';
import 'package:ld_wbench5/utils/debug.dart';

export 'ld_widget_ctrl_abs.dart';
export 'ld_widget_model_abs.dart';

/// Widget base que proporciona funcionalitats comunes
abstract class LdWidgetAbs extends StatefulWidget with LdTaggableMixin {
  /// ID del mapa de configuració
  final String _mapId;

  /// GlobalKey per accedir al controlador
  final GlobalKey<LdWidgetCtrlAbs> _key;

  /// Constructor principal amb mapa de configuració
  LdWidgetAbs({
    super.key,
    required LdMap<dynamic> config,
  }) : _key = GlobalKey<LdWidgetCtrlAbs>(),
        _mapId = MapsService.s.registerMap(
          config,
          pType: "widget",
          pIdent: config[cfTag] as String?,
        ),
        super(key: key ?? GlobalKey()) {
    // Assignar el tag des del mapa de configuració
    tag = MapsService.s.getMap(_mapId)[cfTag] as String? ??
      "LdWidget_${DateTime.now().millisecondsSinceEpoch}";
    Debug.info("$tag: Creant widget amb mapa (ID: $_mapId)");
  }

  /// Constructor alternatiu amb paràmetres individuals
  /// [Obsolet] Utilitzar el constructor amb mapa de configuració
  @deprecated
  LdWidgetAbs.obsolete({
    super.key,
    String? pTag,
    bool isVisible = true,
    bool canFocus = true,
    bool isEnabled = true,
  }) : _key = GlobalKey<LdWidgetCtrlAbs>(),
        _mapId = MapsService.s.registerMap({
          cfTag: pTag ?? "LdWidget_${DateTime.now().millisecondsSinceEpoch}",
          cfIsVisible: isVisible,
          cfCanFocus: canFocus,
          cfIsEnabled: isEnabled,
        }, pType: "widget") {
    tag = pTag ?? "LdWidget_${DateTime.now().millisecondsSinceEpoch}";
    Debug.warn("$tag: Utilitzant constructor obsolet. Canvieu al constructor
amb mapa de configuració.");
  }
}

```

```
/// Allibera recursos quan el widget és eliminat
```

```
@override
```

```
void dispose() {
```

```
  MapsService.s.releaseMap(_mapId);
```

```
  super.dispose();
```

```
}
```

```
/// Obté el mapa de configuració
```

```
LdMap<dynamic> get config => MapsService.s.getMap(_mapId);
```

```
/// Obté la GlobalKey del controlador
```

```
GlobalKey<LdWidgetCtrlAbs> get key => _key;
```

```
@override
```

```
State<LdWidgetAbs> createState();
```

```
}
```

A continuació, la refactorització de `LdWidgetCtrlAbs`:

dart

```
// lib/core/ld_widget/ld_widget_ctrl_abs.dart
```

```
// Controlador del widget base per a l'aplicació.
```

```
// Created: 2025/04/29 dt. CLA[JIQ]
```

```
// Updated: 2025/05/10 ds. CLA - Adaptació per utilitzar MapsService
```

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

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

```
import 'package:ld_wbench5/core/event_bus/event_bus.dart';
```

```
import 'package:ld_wbench5/core/event_bus/ld_event.dart';
```

```
import 'package:ld_wbench5/core/ld_model_abs.dart';
```

```
import 'package:ld_wbench5/core/ld_taggable_mixin.dart';
```

```
import 'package:ld_wbench5/core/ld_widget/ld_widget_abs.dart';
```

```
import 'package:ld_wbench5/core/ld_widget/ld_widget_model_abs.dart';
```

```
import 'package:ld_wbench5/core/lifecycle_interface.dart';
```

```
import 'package:ld_wbench5/core/map_fields.dart';
```

```
import 'package:ld_wbench5/utils/debug.dart';
```

```
/// Controlador base pels widgets.
```

```
abstract class LdWidgetCtrlAbs<T extends LdWidgetAbs>
```

```
  extends State<T>
```

```
  with LdTaggableMixin
```

```
  implements LdLifecycleIntf, LdModelObserverIntf {
```

```
  /// Model del widget
```

```
  LdWidgetModelAbs? _model;
```

```
  /// Retorna el model del widget
```

```
  LdWidgetModelAbs get model => _model!;
```

```

    /// Marca si el model ha estat creat per aquest controlador
    bool _ownsModel = false;

    /// Subscripció als events de l'aplicació
    StreamSubscription<LdEvent>? _subcEvent;

    /// Node de focus pel widget
    final FocusNode _focusNode = FocusNode();

    /// Retorna el node de focus del widget
    FocusNode get focusNode => _focusNode;

    /// PROPIETATS DE CONFIGURACIÓ
    /// Aquestes propietats s'obtenen del mapa de configuració del widget

    /// Flag de visibilitat del widget
    bool _isVisible = true;

    /// Retorna si el widget és visible
    bool get isVisible => _isVisible;

    /// Estableix si el widget és visible i reconstrueix la UI si és necessari
    set isVisible(bool value) {
        if (_isVisible != value && mounted) {
            setState(() {
                _isVisible = value;
                Debug.info("$tag: Visibilitat canviada a $value");
            });
        } else if (_isVisible != value) {
            // Si no està muntat, només canviem el valor
            _isVisible = value;
            Debug.info("$tag: Visibilitat canviada a $value (sense reconstrucció)");
        }
    }

    /// Indica si el widget té el focus
    bool get hasFocus => _focusNode.hasFocus;

    /// Flag que indica si el widget pot rebre focus
    bool _canFocus = true;

    /// Retorna si el widget pot rebre focus
    bool get canFocus => _canFocus;

    /// Estableix si el widget pot rebre focus i reconstrueix la UI si és necessari
    set canFocus(bool value) {
        if (_canFocus != value && mounted) {
            setState(() {

```

```

        _canFocus = value;
        if (!value && hasFocus) {
            _focusNode.unfocus();
        }
        Debug.info("$tag: Capacitat de focus canviada a $value");
    });
} else if (_canFocus != value) {
    // Si no està muntat, només canviem el valor
    _canFocus = value;
    if (!value && hasFocus) {
        _focusNode.unfocus();
    }
    Debug.info("$tag: Capacitat de focus canviada a $value (sense
reconstrucció)");
}
}

/// Flag que indica si el widget està actiu (enabled)
bool _isEnabled = true;

/// Retorna si el widget està actiu
bool get isEnabled => _isEnabled;

/// Estableix si el widget està actiu i reconstrueix la UI si és necessari
set isEnabled(bool value) {
    if (_isEnabled != value && mounted) {
        setState(() {
            _isEnabled = value;
            if (!value && hasFocus) {
                _focusNode.unfocus();
            }
        });
        Debug.info("$tag: Estat d'activació canviat a $value");
    }
} else if (_isEnabled != value) {
    // Si no està muntat, només canviem el valor
    _isEnabled = value;
    if (!value && hasFocus) {
        _focusNode.unfocus();
    }
    Debug.info("$tag: Estat d'activació canviat a $value (sense
reconstrucció)");
}
}

/// Inicialitza el controlador
@override
void initState() {
    super.initState();

    // Assignar un tag basat en el tag del widget

```

```

tag = '${widget.tag}_Ctrl';

// Carregar configuració del widget
_loadConfigFromWidget();

// Crear el model si no existeix
_createModelIfNeeded();

// Subscriure's als events
_subcEvent = EventBus.s.listen(_handleEvent);

// Inicialitzar
initialize();

Debug.info("$tag: Controlador inicialitzat");
}

/// Carrega la configuració des del widget
void _loadConfigFromWidget() {
    final config = widget.config;

    // Carregar propietats del controlador
    _isVisible = config[cfIsVisible] as bool ?? true;
    _canFocus = config[cfCanFocus] as bool ?? true;
    _isEnabled = config[cfIsEnabled] as bool ?? true;

    Debug.info("$tag: Configuració carregada: visible=$_isVisible,
canFocus=$_canFocus, enabled=$_isEnabled");
}

/// Crea el model del widget si és necessari
void _createModelIfNeeded() {
    // En la implementació base, aquest mètode ha de ser sobreescrit
    // per les subclasses per crear un model específic
}

/// Processa un event rebut
void _handleEvent(LdEvent event) {
    // Processar events globals o dirigits a aquest controlador
    if (event.eType == EventType.languageChanged ||
        event.eType == EventType.themeChanged ||
        event.eType == EventType.rebuildUI ||
        event.isTargetedAt(tag)) {
        Debug.info("$tag: Processant event ${event.eType.name}");
        onEvent(event);
    }
}

/// Mètode a sobre escriure per gestionar events
void onEvent(LdEvent event);

```



```

    /// Actualitza el controlador quan canvien les dependències
    @override
    void didChangeDependencies() {
        super.didChangeDependencies();
        Debug.info("$tag: Dependències actualitzades");
        update();
    }

    /// Gestiona l'actualització del widget quan canvien les propietats
    @override
    void didUpdateWidget(covariant T oldWidget) {
        super.didUpdateWidget(oldWidget);

        // Verificar si la configuració ha canviat
        if (_hasConfigChanged(oldWidget)) {
            Debug.info("$tag: Configuració actualitzada, recarregant");
            _loadConfigFromWidget();

            // Forçar reconstrucció si és necessari
            setState(() {
                // Reconstruir la UI
            });
        }
    }

    /// Comprova si la configuració ha canviat
    bool _hasConfigChanged(T oldWidget) {
        // Comparar només les propietats del controlador (cf*)
        final oldConfig = oldWidget.config;
        final newConfig = widget.config;

        // Filtrar i comparar només les propietats de controlador (cf*)
        bool hasChanged = false;
        for (final key in newConfig.keys) {
            if (key.startsWith('cf') && key != cfTag) {
                if (oldConfig[key] != newConfig[key]) {
                    Debug.info("$tag: Canvi detectat en propietat $key: ${oldConfig[key]}
-> ${newConfig[key]}");
                    hasChanged = true;
                    break;
                }
            }
        }

        return hasChanged;
    }

    /// Allibera els recursos
    @override

```

```

void dispose() {
    Debug.info("$tag: Alliberant recursos...");

    // Cancel·lar subscripció a events
    EventBus.s.cancel(_subcEvent);

    // Alliberar recursos del node de focus
    _focusNode.dispose();

    // Alliberar el model si és propietat d'aquest controlador
    if (_ownsModel && _model != null) {
        _model!.dispose();
        _model = null;
    }

    super.dispose();
    Debug.info("$tag: Recursos alliberats");
}

/// Notificació de canvi en un model
@override
void onModelChanged(LdModelAbs pModel, void Function() pfUpdate) {
    Debug.info("$tag: Model ha canviat");

    // Executar la funció d'actualització
    pfUpdate();

    // Reconstruir si està muntat
    if (mounted) {
        setState(() {
            Debug.info("$tag: Reconstruint després del canvi del model");
        });
    }
}

/// Construeix el widget
@override
Widget build(BuildContext context) {
    Debug.info("$tag: Construint widget. isVisible=$_isVisible");

    // Si el widget no és visible, retornem un SizedBox buit
    if (!_isVisible) {
        return const SizedBox.shrink();
    }

    // Construïm el contingut del widget
    Widget content = buildContent(context);

    // Apliquem l'opacitat i l'absorció d'events si el widget no està actiu
    if (!_isEnabled) {

```

```

        content = Opacity(
          opacity: 0.5,
          child: AbsorbPointer(
            absorbing: true,
            child: content,
          ),
        );
      }
    }
    return content;
  }

  /// Mètode que ha d'implementar cada widget per construir la seva UI
  Widget buildContent(BuildContext context);

  /// Demana focus per aquest widget si pot rebre'l i està actiu
  void requestFocus() {
    if (canFocus && isEnabled && isVisible) {
      _focusNode.requestFocus();
      Debug.info("$tag: Focus demanat");
    }
  }

  /// Crea un clon d'aquest controlador
  /// Utilitzat en el mètode createState() del widget
  LdWidgetCtrlAbs<T> clone();
}

```

Finalment, la implementació de `LdWidgetModelAbs` també s'ha d'adaptar:

dart

```

// lib/core/ld_widget/ld_widget_model_abs.dart
// Model base pels widgets.
// Created: 2025/05/02 dj. JIQ
// Updated: 2025/05/10 ds. CLA - Adaptació per usar mapes de configuració

import 'package:ld_wbench5/core/ld_model_abs.dart';
import 'package:ld_wbench5/core/ld_widget/ld_widget_abs.dart';
import 'package:ld_wbench5/core/map_fields.dart';
import 'package:ld_wbench5/ui/extensions/map_extensions.dart';
import 'package:ld_wbench5/utils/debug.dart';

/// Model base pels widgets.
abstract class LdWidgetModelAbs<T extends LdWidgetAbs> extends LdModelAbs {
  /// Construeix un model a partir d'un mapa de propietats
  LdWidgetModelAbs.fromMap(LdMap<dynamic> pMap) {
    // Només assignar les propietats que comencen amb 'mf'
    LdMap<dynamic> modelProperties = {};
    for (var entry in pMap.entries) {
      if (entry.key.startsWith('mf')) {

```

```

        modelProperties[entry.key] = entry.value;
    }
}

// Establir el tag
tag = pMap[mfTag] as String? ?? pMap[cfTag] as String? ?? "Model_${
DateTime.now().millisecondsSinceEpoch}";

// Carregar les propietats
fromMap(modelProperties);

Debug.info("$tag: Model creat a partir de mapa");
}

/// Constructor obsolet
@deprecated
LdWidgetModelAbs(T pwidget) {
    Debug.warn("$tag: Utilitzant constructor obsolet. Canvieu al constructor
amb mapa de propietats.");
}

/// Converteix el model a un mapa de propietats
@override
LdMap<dynamic> toMap() {
    LdMap<dynamic> map = super.toMap();
    map[mfTag] = tag;
    return map;
}

/// Carrega el model des d'un mapa de propietats
@override
void fromMap(LdMap<dynamic> pMap) {
    super.fromMap(pMap);
    // Específic de cada implementació
}

/// Actualitza només les propietats específiques del model
void updateFromMap(LdMap<dynamic> pMap) {
    // Filtrar només les propietats que comencen amb 'mf'
    LdMap<dynamic> modelProperties = {};
    for (var entry in pMap.entries) {
        if (entry.key.startsWith('mf')) {
            modelProperties[entry.key] = entry.value;
        }
    }
}

// Actualitzar només si hi ha propietats
if (modelProperties.isNotEmpty) {
    fromMap(modelProperties);
}

```

```
}  
}
```

Amb aquestes classes base, ara hauríem d'adaptar un dels widgets existents per seguir aquest nou patró. Vaig a actualitzar `LdTextField` per mostrar com quedaria:

dart

```
// lib/ui/widgets/ld_text_field/ld_text_field.dart  
// Widget per a l'edició de text amb suport per a internacionalització.  
// Created: 2025/05/06 dt. CLA  
// Updated: 2025/05/10 ds. CLA - Adaptació a la nova arquitectura amb  
// MapService  
  
import 'package:flutter/material.dart';  
import 'package:ld_wbench5/core/ld_widget/ld_widget_abs.dart';  
import 'package:ld_wbench5/core/map_fields.dart';  
import 'package:ld_wbench5/ui/widgets/ld_text_field/ld_text_field_ctrl.dart';  
import 'package:ld_wbench5/ui/widgets/ld_text_field/ld_text_field_model.dart';  
import 'package:ld_wbench5/utils/debug.dart';  
import 'package:ld_wbench5/ui/extensions/map_extensions.dart';  
  
export 'ld_text_field_ctrl.dart';  
export 'ld_text_field_model.dart';  
  
/// Widget per a l'edició de text amb suport per a internacionalització  
class LdTextField extends LdWidgetAbs {  
  /// Constructor  
  LdTextField({  
    super.key,  
    String? pTag,  
    String initialText = "",  
    String? label,  
    String? helpText,  
    String? errorMessage,  
    bool hasError = false,  
    bool allowNull = true,  
    Function(String)? onTextChanged,  
    bool canFocus = true,  
    bool isEnabled = true,  
    bool isVisible = true,  
  }) : super(config: {  
    // Propietats d'identificació  
    cfTag: pTag ?? "LdTextField_${DateTime.now().millisecondsSinceEpoch}",  
    // Propietats del controlador (cf)  
    cfIsVisible: isVisible,  
    cfCanFocus: canF
```

Retry

Claude hit the max length for a message and has paused its response. You can write Continue to keep the chat going.

JI

Just quan ens van tallar la conversa anterior t'anava a preguntar per la necessitat i utilitat del concepte del camp 'tag'.

El 'tag' era un mecanisme de gestió necessari per a la manipulació de controladors de GetX. Això ens permetia actualitzar els elements visuals (GetBuilder's) que anàvem creant.

Actualment es manté la creació de la clau 'tag' però no estem enregistrant les instàncies enlloc.

Analitza la necessitat de continuar mantenint la creació dels tags i de si estem solapant serveis.

Edit

Tens raó en qüestionar la necessitat i utilitat del camp 'tag' en la nostra arquitectura actual. Anem a analitzar-ho en profunditat:

Anàlisi del concepte 'tag'

Origen i ús inicial:

- Com bé indiqueu, el 'tag' era un mecanisme de gestió per a manipular controladors en el context de GetX.
- GetX utilitza tags per identificar i accedir a controladors registrats globalment.
- Permetia actualitzar els GetBuilder amb un identificador únic.

Situació actual al codi:

- Estem mantenint el `LdTaggableMixin` que proporciona funcionalitat de tag a diverses classes.
- Els tags s'utilitzen principalment per:

1. Debugging (als logs mitjançant `Debug.info("$tag: ...")`)
2. Identificació de components en events (en `EventBus` per enviar events específics)
3. Identificació en el `MapsService` com a potencial identificador del mapa

Solapament amb altres serveis:

- **Keys de Flutter:** Flutter ja proporciona `Key` i `GlobalKey` per identificar i accedir a widgets i states.

- **MapId a MapsService:** Ja tenim un sistema d'identificadors (`_mapId`) per als mapes al `MapsService`.
- **EventBus:** Utilitza tags per dirigir events a components específics.

Anàlisi crítica

Pros de mantenir els tags:

1. **Debugging:** Proporciona noms significatius als logs, facilitant la depuració.
2. **Events específics:** Permet enviar events a components específics sense necessitat de `GlobalKeys`.
3. **Familiaritat:** L'equip ja està acostumat a treballar amb aquest concepte.

Contres de mantenir els tags:

1. **Redundància:** Solapa parcialment amb el sistema de keys de Flutter.
2. **Complexitat addicional:** Afegeix un concepte més a gestionar.
3. **No centralitzat:** No hi ha un registre central de tags, cosa que pot portar a duplicitats.
4. **Overhead de memòria:** Tot i ser petit, cada string és memòria addicional.

Recomanacions

Basant-me en l'anàlisi, proposo aquestes alternatives:

Opció 1: Mantenir tags només per a debugging

- Simplificar `LdTaggableMixin` per només generar tags descriptius per a logs.
- No utilitzar els tags per a cap lògica funcional.
- Utilitzar `GlobalKeys` de Flutter per a funcionalitat d'accés.

Opció 2: Integrar tags amb MapsService

- Unificar el concepte de tag amb l'ID del mapa a `MapsService`.
- Utilitzar el `mapId` com a identificador únic a tota l'aplicació.
- Modificar `EventBus` per utilitzar `mapIds` en lloc de tags.

Opció 3: Crear un registre centralitzat

- Crear un `TagRegistry` que gestioni tots els tags i eviti duplicitats.
- Proporcionar funcionalitat de recerca per tag.
- Integrar-lo amb `MapsService` per unificar la gestió d'identificadors.

Recomanació final

Recomano l'**Opció 2**: Integrar els tags amb MapsService, perquè:

1. Elimina la redundància.
2. Centralitza la gestió d'identificadors.
3. Manté la funcionalitat actual de tags sense complexitat addicional.
4. És coherent amb l'arquitectura basada en mapes que estem implementant.

Implementació:

- Modificar `LdTaggableMixin` per utilitzar el `mapId` com a tag.
- Afegir un mètode a `MapsService` per obtenir un tag descriptiu a partir d'un `mapId`.
- Actualitzar `EventBus` per utilitzar `mapIds` en lloc de tags.

Què en penses d'aquestes propostes? Creus que alguna d'aquestes opcions és viable o tens alguna altra idea per abordar aquest tema?

Retry

JI

A veure, fem una cosa. Què necessitem enregistrar per a accedir amb clau?: 1. Hem de poder actualitzar qualsevol element visual que hi hagi en pantalla. És a dir widgets i per tant `'GlobalKey<WidgetImpl>()'`. 2. Això també implica poder enviar missatges específics (events) a oïdors. Això ho fem fins ara amb 'tag', però ho fem implementant `LdModelObserver`. 3. També necessitem enregistrar mapes de dades per tal que els estats (controladors) puguin treballar amb una única còpia de les dades a part de poder fer servir el mecanisme de registre de mapes a la resta de la lògica de l'aplicació que no necessàriament tindrà impacte visual (lògica de negoci en segon pla).

Fem una cosa, escriu una llista amb els conceptes pels quals necessitem guardar una referència i una còpia única en algun registre. Identifica per a cada tipus de concepte quina clau fem servir fins ara i quina seria més lògica. Tingues present que no només estem parlant de widgets o objectes visuals. L'aplicació tindrà una capa de negoci que també em de tenir present.

Edit

Anàlisi dels conceptes que requereixen registre i accés per clau

Concepte	Descripció	Clau actual	Clau proposada	Justificació
Widgets	Elements	tag + <code>GlobalKey</code>	<code>GlobalKey<LdWidget</code>	Flutter ja proporciona

Concepte	Descripció	Clau actual	Clau proposada	Justificació
StatefulWidget	visuals de la interfície d'usuari	<LdWidgetCtrlAbs>	CtrlAbs>	GlobalKey que identifica unívocament widgets i permet accedir al seu State (controlador)
States (controladors)	Gestionen el cicle de vida, estat i comportament dels widgets	tag (derivat del widget)	GlobalKey<LdWidgetCtrlAbs> del widget pare	Els States són accessibles a través de la GlobalKey del seu widget pare amb key.currentState
Models de Widget	Contenen les dades específiques d'un widget	Sense registre centralitzat, accés via el State	Sense necessitat de registre centralitzat	Els models poden ser propietat del State (controlador) sense necessitat d'accés extern directe
Mapes de configuració de Widget	Configuració dels widgets	_mapId a MapsService	_mapId a MapsService	El sistema actual amb MapsService és adequat per gestionar configuracions compartides
Models de negoci	Entitats i lògica de negoci independent de la UI	Sense registre centralitzat, accés directe	id únic + registre a un EntityRegistry	Els models de negoci requereixen identificadors únics que persisteixin entre sessions
Subscriptors d'events	Components que necessiten ser notificats d'events	tag a EventBus	id únic consistent amb altres registres	Unificar tots els identificadors per facilitar l'integració de sistemes
Serveis de l'aplicació	Lògica global compartida (auth, API, etc.)	Singletons amb nom de classe	Pattern Singleton + nom descriptiu	Els serveis són globals i únics per definició, un registre explícit seria redundant
Recursos	Assets, imatges, textos, etc.	Literal strings	Enums o constants	Els recursos són estàtics i predefinits, més adequat utilitzar enums o constants
Sessions d'usuari	Informació de l'usuari actual	No implementat	Token únic + SessionManager	La informació d'usuari ha de ser accessible globalment però segura
Cau de dades	Dades en memòria per reduir càrrega de xarxa	No implementat	Clau basada en contingut (hash)	Les claus basades en contingut permeten identificar unívocament les dades independentment de l'origen

Anàlisi i recomanacions

Per a elements d'interfície d'usuari:

- **Clau recomanada:** GlobalKey<T> on T és el tipus específic de State
- **Justificació:**

1. Flutter ja proporciona aquest mecanisme, és estàndard i robust
2. Permet accés tipat i segur a l'estat del widget
3. Es pot passar entre components sense necessitat d'un registre global
4. Exemple d'ús: `myKey.currentState?.method()`

Per a mapes de configuració:

- **Clau recomanada:** `_mapId` gestionat per `MapsService`
- **Justificació:**

1. Sistema ja implementat i funcional
2. Permet compartir configuracions entre instàncies
3. Gestiona automàticament el comptador de referències i neteges
4. Exemple d'ús: `MapsService.s.getMap(_mapId)`

Per a models de negoci:

- **Clau recomanada:** `EntityId<T>` (personalitzat per tipus d'entitat)
- **Justificació:**

1. Els models de negoci requereixen identificadors persistents
2. Diferent dels identificadors d'UI efímers
3. Han de ser compatibles amb backend/API
4. Exemple d'ús: `EntityRegistry.get<User>(userId)`

Per a comunicació d'events:

- **Clau recomanada:** Identificador unificat basat en la natura del component
- **Justificació:**

1. Per widgets: utilitzar el mateix identificador que el widget (`GlobalKey` o `_mapId`)
2. Per serveis: utilitzar nom de classe o identificador de servei
3. Per models de negoci: utilitzar `EntityId`
4. Exemple d'ús: `EventBus.s.emit(event, targetId: widget.key)`

Proposta unificada

Per simplificar i unificar el sistema, proposo:

1. Classe `SystemId`:

- Classe abstracta base per tots els identificadors del sistema
- Subtipus específics: `WidgetId`, `ModelId`, `ServiceId`, etc.

- Mètode toString() per debugging
- Mètode getRegistryType() per saber quin registre utilitzar

2. Classe **SystemRegistry**:

- Singleton amb accés global
- Mètodes register(SystemId, Object)
- Mètodes get<T>(SystemId)
- Suport per comptadors de referències
- Integració amb cicle de vida per netejar automàticament

3. Integració amb sistemes existents:

- MapsService esdevindria una especialització de SystemRegistry
- EventBus utilitzaria SystemId per dirigir events
- Flutter GlobalKey s'encapsularien dins WidgetId

Això permetria tenir un sistema coherent i unificat d'identificació i accés als diferents components de l'aplicació, des de la interfície fins a la lògica de negoci.

Què en penses d'aquesta anàlisi i proposta? Creus que és massa complex o, al contrari, que aborda adequadament les necessitats del projecte?

Retry

JI

Hem de aclarir la diferència entre l'identificador d'un model que conté una instància d'una entitat de base de dades, on hi haurà un camp propi de clau assistida pel sistema de BD; l'identificador d'un objecte visual (widget, controlador) i l'identificador del model d'un d'aquests widgets.

El model d'un widget (sigui un component o una pàgina sencera) és un mapa especialitzat pel tipus de widget, no té una relació necessàriament forta amb cap entitat de dades de l'aplicació. És un model orientat a permetre la gestió visual de dades a través de widgets segons les necessitats.

El model d'una entitat de base de dades és un mapa que reflexa fidedignament 1 a 1 el contingut d'un registre d'una o varies taules de base de dades.

El model d'informació que transporta un event és el més efímer, està ideat per a consumir i llençar. Pot ser qualsevol agrupació d'elements d'un mapa.

La comunicació mitjançant proto3 també haurà de fer servir un model d'informació que després poguem traduir amb facilitat al servidor i a l'aplicació.

La idea de 'LdModelAbs' és abstraure totes aquests tipus d'estructures en un concepte únic.

Tant 'LdWidgetAbs' com 'LdModelAbs' i 'LdPageAbs' ja disposen de tag a través del mixin 'LdTaggableMixin'. Per tant estem barrejant naps amb cols.

Parla'm més sobre GlobalKey. Ha d'estar relacionat necessàriament amb objectes visuals o és una abstracció general de clau?

Edit

Tens raó, és important aclarir aquestes diferències. Anem a examinar més a fons cada tipus d'identificador i explicar millor què és GlobalKey i les seves limitacions.

GlobalKey a Flutter

GlobalKey a Flutter és un tipus d'identificador específicament dissenyat per a elements de la interfície d'usuari (widgets). No és una abstracció general de clau que pugui utilitzar-se per a qualsevol tipus d'objecte.

Característiques principals de GlobalKey:

1. **Específic per widgets:** Només funciona amb objectes que són subclasses de widget.
2. **Accés al State:** Permet accedir a l'objecte State associat a un StatefulWidget.
3. **Accés al BuildContext:** Proporciona accés al BuildContext del widget.
4. **Navegació:** Permet accedir a elements de widgets d'arbre de components.
5. **Tipus genèric:** És possible especificar el tipus de State per obtenir accés tipat: GlobalKey<MyWidgetState>.

Limitacions importants:

1. **No es pot utilitzar per identificar models de negoci** o qualsevol objecte que no sigui un widget.
2. **Overhead de rendiment:** L'ús excessiu de GlobalKey pot tenir impacte en el rendiment.
3. **Cicle de vida lligat a la UI:** Quan el widget s'elimina, la referència a través de GlobalKey deixa de ser vàlida.

Per tant, GlobalKey és adequat només per a la identificació i accés a elements visuals, però no per a models de dades, entitats de negoci, o altres objectes de l'aplicació.

Aclarint els diferents tipus de models i identificadors

1. Model d'un widget (LdWidgetModelAbs)

- **Propòsit:** Gestionar l'estat i comportament visual d'un component específic.
- **Durada:** Existeix mentre el widget és visible o està en memòria.

- **Identificació actual:** tag heretat de `LdTaggableMixin`.
- **Relació amb dades:** Pot contenir dades transients o referències a models de negoci, però no necessàriament representa una entitat completa.

2. Model d'entitat de negoci

- **Propòsit:** Representar una entitat de les dades del negoci (usuari, producte, etc.).
- **Durada:** Pot persistir independentment de la UI, sovint amb cicle de vida llarg.
- **Identificació:** Necessita un identificador únic persistent (sovint provinent de la base de dades).
- **Relació amb dades:** Mapeja directament 1:1 amb un o més registres de la base de dades.

3. Model d'informació d'event

- **Propòsit:** Transportar informació específica sobre un esdeveniment.
- **Durada:** Efímera, només durant el procés de notificació.
- **Identificació:** Normalment no necessita identificador propi, però sí un tipus d'esdeveniment (`EventType`).
- **Relació amb dades:** Conté només la informació necessària per gestionar l'esdeveniment.

4. Model de comunicació proto3

- **Propòsit:** Serialitzar i deserialitzar dades per a comunicació entre client i servidor.
- **Durada:** Temporal, només durant la transmissió.
- **Identificació:** Definit pel protocol (proto3) mitjançant tipus de missatge.
- **Relació amb dades:** Estructura estrictament definida pel schema proto3.

Proposta refinada per a identificació

Donat aquest aclariment, crec que necessitem un enfocament més específic per a cada tipus d'objecte:

Per a widgets i els seus controladors:

- Mantenir tag a través de `LdTaggableMixin` **només per a propòsits de debugging.**
- Afegir `GlobalKey<T>` per a accés programàtic des d'altres parts de l'aplicació.
- Exemple:

```
final buttonKey = GlobalKey<LdButtonCtrl>(); ...
buttonKey.currentState?.toggleEnabled();
```

Per a mapes de configuració de widgets:

- Continuar utilitzant `MapService` amb `_mapId` com a identificador.
- Aquest identificador és intern i no hauria d'exposar-se fora de la implementació.

Per a models d'entitats de negoci:

- Implementar un sistema d'identificadors específic per a entitats: `EntityId`.
- Aquest identificador hauria d'encapsular l'ID de la base de dades i el tipus d'entitat.
- Exemple: `EntityId(type: "User", id: 123)`

Per a comunicació d'events:

- Utilitzar `EventType` per identificar el tipus d'event.
- Utilitzar `srcTag` per identificar l'emissor (ja implementat).
- Utilitzar `tgtTags` per identificar els receptors (ja implementat).

Per a comunicació proto3:

- Utilitzar els tipus de missatge definits al schema proto3.
- No cal un sistema d'identificació addicional, ja que proto3 defineix l'estructura.

Simplificació de `LdTaggableMixin`

Donat que el `tag` s'utilitza principalment per a propòsits de debugging i no per a funcionalitat crítica, podríem simplificar-lo:

`dart`

```
/// Mixin que proporciona capacitats d'identificació per a debugging
mixin LdTaggableMixin {
  /// Nom base de la classe
  String get className => runtimeType.toString().replaceAll(RegExp(r'<.*>'),
    '');

  /// Tag únic per a aquesta instància (només per debugging)
  String? _tag;

  /// Obté el tag d'aquesta instància, generant-ne un si no existeix
  String get tag => _tag ?? generateTag();

  /// Estableix el tag
  set tag(String pNewTag) {
    _tag = pNewTag;
    Debug.info("Tag assignat: $_tag");
  }
}
```

```
/// Genera un tag únic basat en el tipus de classe i timestamp
String generateTag() {
  final generatedTag = '${className}_${
DateTime.now().millisecondsSinceEpoch}';
  _tag = generatedTag;
  return generatedTag;
}
```

Aquesta proposta aborda millor les diferents necessitats d'identificació segons el tipus d'objecte, mantenint la compatibilitat amb el codi existent mentre proporciona un camí clar per a millores futures.

Què en penses d'aquesta anàlisi més detallada? És més proper al que tenies en ment?

Retry

JI

Explica'm el cicle de vida d'una 'GlobalKey'.

Edit

El cicle de vida d'una GlobalKey a Flutter està estretament lligat al cicle de vida dels widgets amb els quals està associada. Anem a examinar-lo pas a pas:

Cicle de vida d'una GlobalKey

1. Creació

dart

```
// Creació de la GlobalKey
final myWidgetKey = GlobalKey<MyWidgetState>();

// La key es pot crear en diferents moments:
// - Com a camp d'una classe (persistent)
class MyPage extends StatelessWidget {
  final GlobalKey<FormState> formKey = GlobalKey<FormState>();
  // ...
}

// - Localment en un mètode (temporal)
void createWidget() {
  final localKey = GlobalKey<MyWidgetState>();
  // ...
}
```

2. Assignació al widget

dart

```
MyWidget(  
  key: myWidgetKey, // Assignem la key al widget  
  // ... altres propietats  
)
```

3. Muntatge (Mount)

- Quan el widget es crea i s'afegeix a l'arbre de widgets
- La key `isForwarded = true` i es vincula amb el widget
- `key.currentState` encara és `null` fins que el widget estigui completament muntat

4. Post-build / Inicialització completa

- Després que el widget ha completat `initState()` o `build()`
- Ara `key.currentState` retorna la instància del State associat
- `key.currentContext` retorna el BuildContext del widget

5. Fase activa

dart

```
// Durant aquesta fase pots accedir a:  
myWidgetKey.currentState?.someMethod(); // Crida mètodes del State  
myWidgetKey.currentContext;              // Obté el BuildContext  
myWidgetKey.currentWidget;               // Obté el widget actual
```

6. Reconstruccions (Rebuilds)

- La key persisteix durant les reconstruccions del widget
- `currentState` i `currentContext` es mantenen vàlids
- Això permet mantenir referència al mateix State fins i tot quan el widget es reconstrueix

7. Desmuntatge (Unmount)

dart

```
// Quan el widget es retira de l'arbre de widgets:  
// - currentState torna a ser null  
// - currentContext torna a ser null  
// - La key encara existeix però no està vinculada a cap widget
```


8. Alliberament

dart

```
// La key pot ser reassignada a un nou widget
MyAnotherWidget(
  key: myWidgetKey, // Reutilitzem la mateixa key
)

// O es pot perdre tota referència
myWidgetKey = null; // La key quedarà recollida pel GC
```

Característiques importants

Persistència entre reconstruccions

dart

```
class ParentWidget extends StatefulWidget {
  const ParentWidget({Key? key}) : super(key: key);
}

@override
State<ParentWidget> createState() => _ParentWidgetState();
}

class _ParentWidgetState extends State<ParentWidget> {
  final childKey = GlobalKey<ChildWidgetState>();
  bool showChild = true;
}

@override
Widget build(BuildContext context) {
  return Column(
    children: [
      if (showChild)
        ChildWidget(key: childKey),
      ElevatedButton(
        onPressed: () {
          setState(() {
            showChild = !showChild;
          });
        },
        child: Text('Toggle Child'),
      ),
    ],
  );
}
}
```

En aquest exemple:

- Quan `showChild` passa de `true` a `false`, el widget es desmunta
- Quan torna a `true`, s'utilitza la mateixa key
- Si `ChildWidget` és un `StatefulWidget`, mantindrà el seu estat

Detecció d'estat

dart

```
// Pots comprovar si la key està activa:
if (myWidgetKey.currentState != null) {
  // El widget està muntat i accessible
  myWidgetKey.currentState?.doSomething();
} else {
  // El widget no està muntat o ha sigut eliminat
}
```

Errorrs comuns

dart

```
// Error 1: Duplicar la mateixa key
Widget build(BuildContext context) {
  return Column(
    children: [
      Widget(key: myKey), // Error!
      Widget(key: myKey), // Mateixa key utilitzada dues vegades
    ],
  );
}
```

```
// Error 2: Accedir abans del muntatge
void createWidget() {
  final key = GlobalKey();
  Widget(key: key);
}

// Error - el widget encara no està muntat
key.currentState?.doSomething();

// Correcte - esperar que estigui muntat
WidgetsBinding.instance.addPostFrameCallback((_) {
  key.currentState?.doSomething();
});
}
```

Millors pràctiques

1. **Crear keys com a membres de classe** quan necessitis accés persistent
2. **No crear keys innecessàriament** - tenen overhead de memòria

3. **Comprovar sempre** si `currentState` és `null` abans d'usar-lo
4. **Reutilitzar keys amb precaució** - només si realment vols mantenir l'estat
5. **Evitar globalkeys** per a widgets estàtics - usar keys simples en aquests casos

Aquesta és la dinàmica completa del cicle de vida d'una `GlobalKey`. És especialment útil quan necessites accedir a un widget específic des d'una altra part de l'aplicació, però cal anar amb compte amb el timing i la gestió de memòria.

Retry

JI

D'acord. Llavors una altra pregunta. Veig que les keys es propaguen fins a l'arrel dels widgets a través del constructor (`super(key: key)`). És a dir que és el widget especialitzat el responsable de crear la seva clau i de transmetre-la fins al seu ancestre major (`StatefulWidget`).

És a dir, que com a mecànica de creació té un funcionament equivalent al que nosaltres fem servir amb 'tag', que també podem traslladar-lo des del widget més especialitzat fins a 'LdWidgetAbs' o 'LdPageAbs', però que si no s'especifica es crea en la classe base a partir del nom específic de cada classe fina.

Fins aquí és correcte, oi?

Edit

Exactament! Has entès perfectament el funcionament. La mecànica de propagació de `key` i `tag` és molt similar. Deixa'm confirmar i aclarir els punts que has mencionat:

Mecànica de propagació de key

Sí, és correcte. La `key` es propaga des del widget especialitzat fins a l'arrel:

dart

```
// Widget especialitzat
LdTextField(
  key: myTextFieldKey, // Definida aquí
  label: "Nom"
)
```

↓ passa la key a través del constructor

```
// Widget base
LdWidgetAbs({
  super.key, // Rep la key i la propaga
  required Map<String, dynamic> config,
}) : super(key: key) // La passa al StatefulWidget
```

↓ continua propagant

```
// Finalment arriba al StatefulWidget
class LdWidgetAbs extends StatefulWidget {
  LdWidgetAbs({Key? key}) : super(key: key);
}
```

Comparació entre key i tag

La comparació que fas és correcta i molt perspicaç:

Característica	key (Flutter)	tag (nostre)
Creació	Al widget especialitzat o base	Al widget especialitzat o base
Propagació	Via constructors amb <code>super(key: key)</code>	Via <code>tag =</code> en cada classe
Valor per defecte	<code>null</code> si no s'especifica	Auto-generat a la classe base
Ús principal	Identificar widgets per a accés programàtic	Debugging i identificació
Tipus	Objecte (Key)	String

Exemple pràctic amb tag:

dart

```
// 1. Crear un widget amb tag específic
LdTextField(
  pTag: "email_field", // Específicament definit
  label: "Email"
)

// 2. 0 sense tag específic
LdTextField(
  // No especifiquem pTag
  label: "Password"
)

// A la classe base LdWidgetAbs:
LdWidgetAbs({
  String? pTag, // Rep el tag opcional
  // ...
}) {
  // Si no hi ha tag, en genera un
  tag = pTag ?? "LdTextField_${DateTime.now().millisecondsSinceEpoch}";
}
```

Diferències importants

Tot i que el funcionament és similar, hi ha algunes diferències clau:

1. **Flutter key és obligatòria en el sistema:** Tot widget té una propietat `key` (que pot ser `null`).
2. **Nostre tag és opcional:** És un mixin que podem afegir o no.
3. **key té funcionalitat integrada:** Flutter l'utilitza internament per a identificar widgets.
4. **tag és per a ús extern:** Nosaltres el fem servir principalment per debugging i events.

Proposta de millora

Basant-nos en aquesta anàlisi, podríem millorar la nostra implementació:

dart

```
// Opció 1: Utilitzar key com a base per a tag
abstract class LdWidgetAbs extends StatefulWidget with LdTaggableMixin {
  LdWidgetAbs({
    super.key,
    String? pTag,
    // ... altres propietats
  }) {
    // Si no hi ha tag específic, usar el hash de la key si existeix
    if (pTag == null && key != null) {
      tag = "${className}_${key.hashCode}";
    } else {
      tag = pTag ?? "${className}_${DateTime.now().millisecondsSinceEpoch}";
    }
  }
}
```

```
// Opció 2: Crear un sistema unificat
class WidgetIdentifier {
  final Key? key;
  final String? tag;

  WidgetIdentifier({this.key, this.tag});

  String get effectiveTag => tag ?? "Widget_${key?.hashCode ??
DateTime.now().millisecondsSinceEpoch}";
}
```

Així que sí, la teva comprensió és absolutament correcta. La mecànica és similar, i això ens permet tenir un sistema coherent d'identificació que combina el millor dels dos móns.

Retry

JI

Tots els objectes necessiten un 'tag' però només els objectes visuals o widgets necessiten un 'GlobalKey'. Això em fa pensar en una proposta que vas fes anteriorment però modificada: 1.

Incorporem una 'GlobalKey<WidgetState>?' opcional en el mixin 'LdTaggableMixin'. 2. El mixin donarà accés tant a la key opcional com a les eines pròpies de la key si s'ha establert. 3. Donat que el tag s'actualitza e la recreació de les instàncies dels objectes que implementen el mixin la GlobalKey també s'inicialitzarà si es tracta d'un widget. 4. La GlobalKey d'un Widget (sigui component o pàgina) es crea i s'obté directament del mixin, quan l'objecte taggat és un component derivat de StatefulWidget. 5. El sistema d'events pot identificar la font i els objectius opcionals dels events directament amb el tag (independentment de si es tracta d'un widget, un servei, una estructura de dades auxiliar, ...). 6. El debug continua fent servir directament el tag establert. 7. La GlobalKey no ens preocuparà a partir d'ara perquè estara gestionada per LdWidgetAbs i LdPageAbs. Si necessitem accedir-hi la trobarem de forma transparent perquè tots els objectes visuals també implementen el LdTaggableMixin.

Creus que és una solució pràctica i suficient?

Edit

Aquesta proposta és molt elegant i pràctica! Combina el millor dels dos móns (tags per a identificació general i GlobalKey per a widgets) de manera transparent. Deixa'm desenvolupar la implementació:

dart

```
// lib/core/ld_taggable_mixin.dart
// Mixin unificat que gestiona tant tag com GlobalKey
// Created: 2025/05/11 ds. CLA

import 'package:flutter/material.dart';
import 'package:ld_wbench5/utils/debug.dart';
import 'package:ld_wbench5/ui/extensions/type_extensions.dart';

/// Mixin que proporciona capacitats d'identificació mitjançant tags únics
/// i GlobalKey opcional per a widgets
mixin LdTaggableMixin {
  /// Nom base de la classe
  String get className => runtimeType.cleanClassName;

  /// Tag únic per a aquesta instància
  String? _tag;

  /// GlobalKey opcional per a widgets
  GlobalKey? _globalKey;

  /// Obté el tag d'aquesta instància, generant-ne un si no existeix
  String get tag => _tag ?? generateTag();

  /// Estableix el tag com una propietat simple
  set tag(String pNewTag) {
```

```

        _tag = pNewTag;
        Debug.info("Tag assignat: $_tag");
    }
}

/// Obté la GlobalKey si està disponible
GlobalKey? get globalKey => _globalKey;

/// Genera un tag únic basat en el tipus de classe i timestamp
String generateTag() {
    final generatedTag = '${className}_${
DateTime.now().millisecondsSinceEpoch}';
    _tag = generatedTag;
    return generatedTag;
}

/// Inicialitza la GlobalKey per a widgets
/// Aquest mètode hauria de ser cridat pels constructors de LdWidgetAbs i
LdPageAbs
void initializeGlobalKey<T extends State>() {
    if (this is StatefulWidget && _globalKey == null) {
        _globalKey = GlobalKey<T>();
        Debug.info("$tag: GlobalKey inicialitzada per a widget");
    }
}

/// Accés tipat a l'estat del widget si està disponible
T? getWidgetState<T extends State>() {
    if (_globalKey != null) {
        try {
            return _globalKey!.currentState as T?;
        } catch (e) {
            Debug.warn("$tag: Error al obtenir estat del widget: $e");
            return null;
        }
    }
    return null;
}

/// Accés al context del widget si està disponible
BuildContext? get context {
    if (_globalKey != null) {
        return _globalKey!.currentContext;
    }
    return null;
}

/// Comprova si el widget està muntat i accessible
bool get isWidgetMounted => _globalKey?.currentState != null;
}

```

Ara actualitzem LdWidgetAbs per fer servir aquesta funcionalitat:

dart

```
// lib/core/ld_widget/ld_widget_abs.dart
// Widget base amb suport per a tag i GlobalKey unificats
// Updated: 2025/05/11 ds. CLA

import 'package:flutter/material.dart';
import 'package:ld_wbench5/core/ld_taggable_mixin.dart';
import 'package:ld_wbench5/core/map_fields.dart';
import 'package:ld_wbench5/services/maps_service.dart';
import 'package:ld_wbench5/ui/extensions/map_extensions.dart';
import 'package:ld_wbench5/utils/debug.dart';

/// Widget base que proporciona funcionalitats comunes
abstract class LdWidgetAbs extends StatefulWidget with LdTaggableMixin {
  /// ID del mapa de configuració
  final String _mapId;

  /// Constructor principal amb mapa de configuració
  LdWidgetAbs({
    Key? key,
    required LdMap<dynamic> config,
  }) : _mapId = MapsService.s.registerMap(
    config,
    pType: "widget",
    pIdent: config[cfTag] as String?,
  ),
    super(key: key) {
    // Assignar el tag des del mapa de configuració
    tag = config[cfTag] as String? ??
      "LdWidget_${DateTime.now().millisecondsSinceEpoch}";

    // Inicialitzar la GlobalKey per a aquest widget
    initializeGlobalKey<State>();

    Debug.info("$tag: Creant widget amb mapa (ID: $_mapId)");
  }

  /// Sobreescrivim key per retornar la nostra GlobalKey
  @override
  Key? get key => globalKey;

  /// Allibera recursos quan el widget és eliminat
  @override
  void dispose() {
    MapsService.s.releaseMap(_mapId);
    super.dispose();
  }
}
```



```

    /// Obté el mapa de configuració
    LdMap<dynamic> get config => MapsService.s.getMap(_mapId);

    @override
    State<LdWidgetAbs> createState();
}

I actualitzem també LdPageAbs:

dart

// lib/core/ld_page/ld_page_abs.dart
// Pàgina base amb suport per a tag i GlobalKey unificats
// Updated: 2025/05/11 ds. CLA

import 'package:flutter/material.dart';
import 'package:ld_wbench5/core/ld_page/ld_page_ctrl.dart';
import 'package:ld_wbench5/core/ld_page/ld_page_model.dart';
import 'package:ld_wbench5/core/ld_taggable_mixin.dart';
import 'package:ld_wbench5/utils/once_set.dart';

/// Pàgina base que proporciona funcionalitats comunes
abstract class LdPageAbs extends StatefulWidget with LdTaggableMixin {
    /// Controlador i model (com abans)
    final OnceSet<LdPageCtrl> _ctrl = OnceSet<LdPageCtrl>();
    LdPageCtrl get vCtrl => _ctrl.get()!;
    set vCtrl(LdPageCtrl pCtrl) => _ctrl.set(pCtrl);

    final OnceSet<LdPageModelAbs> _model = OnceSet<LdPageModelAbs>();
    LdPageModelAbs get vModel => _model.get()!;
    set vModel(LdPageModelAbs pModel) => _model.set(pModel);

    /// Constructor
    LdPageAbs({
        super.key,
        String? pTag,
    }) {
        tag = pTag ?? className;

        // Inicialitzar la GlobalKey per a aquesta pàgina
        initializeGlobalKey<LdPageCtrl>();
    }

    /// Sobreescrivim key per retornar la nostra GlobalKey
    @override
    Key? get key => globalKey;

    @override
    State<LdPageAbs> createState() => _ctrl.get()!;
}

```

Actualització de l'EventBus per fer servir tags:

dart

```
// lib/core/event_bus/event_bus.dart
```

```
// Sistema d'events amb suport per a tags unificats
```

```
// Updated: 2025/05/11 ds. CLA
```

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

```
import 'package:ld_wbench5/core/event_bus/ld_event.dart';
```

```
import 'package:ld_wbench5/utils/debug.dart';
```

```
/// Gestor centralitzat d'events de l'aplicació
```

```
class EventBus {
```

```
  // ... (codi existent)
```

```
  /// Emet un event a un component específic pel seu tag
```

```
  void emitToTag(LdEvent event, String targetTag) {
```

```
    if (!_ctrl.isClosed) {
```

```
      Debug.info("EventBus: Emitting event of type ${event.eType} to tag  
$targetTag");
```

```
      LdEvent targetedEvent = LdEvent(
```

```
        eType: event.eType,
```

```
        srcTag: event.srcTag,
```

```
        eData: event.eData,
```

```
        tgtTags: [targetTag],
```

```
      );
```

```
      _ctrl.add(targetedEvent);
```

```
    }
```

```
  }
```

```
  /// Emet un event a múltiples components pels seus tags
```

```
  void emitToTags(LdEvent event, List<String> targetTags) {
```

```
    if (!_ctrl.isClosed) {
```

```
      Debug.info("EventBus: Emitting event of type ${event.eType} to tags  
$targetTags");
```

```
      LdEvent targetedEvent = LdEvent(
```

```
        eType: event.eType,
```

```
        srcTag: event.srcTag,
```

```
        eData: event.eData,
```

```
        tgtTags: targetTags,
```

```
      );
```

```
      _ctrl.add(targetedEvent);
```

```
    }
```

```
  }
```

```
}
```

Exemple d'ús en un widget:

dart

```
// Exemple d'ús transparent
class MyWidget extends LdWidgetAbs {
  MyWidget({
    super.key,
    String? pTag,
  }) : super(config: {
    cfTag: pTag ?? "MyWidget_${DateTime.now().millisecondsSinceEpoch}",
    // ... altres propietats
  });

  @override
  State<MyWidget> createState() => MyWidgetCtrl();
}

class MyWidgetCtrl extends LdWidgetCtrlAbs<MyWidget> {
  void someMethod() {
    // Accés transparent a altres widgets pel seu tag
    final otherWidget = findWidgetByTag<OtherWidgetCtrl>("other_widget_tag");
    otherWidget?.doSomething();

    // Enviar event a un tag específic
    EventBus.s.emitToTag(
      LdEvent(
        eType: EventType.custom,
        srcTag: tag, // El nostre tag
        eData: {'message': 'Hello!'},
      ),
      "target_widget_tag"
    );
  }
}

// Helper global per trobar widgets per tag
T? findWidgetByTag<T extends State>(String tag) {
  // Implementar una manera de buscar per tag
  // Requereix un registre global de tags o recórrer l'arbre
  // ... implementació
}
```

Avantatges d'aquesta solució:

1. **Unificació:** Un únic sistema per a identificació de tots els objectes
2. **Transparència:** GlobalKey disponible automàticament per a widgets
3. **Simplicitat:** Els desenvolupadors només treballen amb tags
4. **Compatibilitat:** EventBus pot continuar funcionant exactament igual

5. **Debugging:** Tag sempre disponible per a tots els objectes
6. **Accés programàtic:** GlobalKey disponible quan es necessita

Consideracions:

1. **Memòria:** Cada widget tindrà una GlobalKey, fins i tot si no s'utilitza
2. **Registre de tags:** Potser necessitem un registre global de tags a widgets per facilitar la cerca
3. **Tipus de GlobalKey:** Hauríem de considerar si usar GlobalKey generic o tipat

Aquesta solució és elegant, pràctica i manté la simplicitat d'ús mentre proporciona accés a funcionalitats avançades quan calguin. Què en penses?

Retry

JI

Propostes d'optimització: 1. LdWidgetAbs i LdPageAbs ja no reben un paràmetre opcional 'key' perquè el crearem i gestionarem dins d'aquestes classes directament amb el mixin. 2. El 'super(key: key)' seria el punt de creació: 'super(key: initializeGlobalKey<T extends State>())' o des del cos del constructor si hi ha algun problema d'accés previ a elements del component abans de finalitzar la seva creació. 3. Oferim la interfície de GlobalKey a partir del mateix mixin, com has començat a fer amb 'bool get isWidgetMounted'. D'aquesta forma només fem referència a la GlobalKey quan és realment necessària i deixem d'estendre la necessitat d'incorporar un camp 'key' als constructors superiors.

Edit

Excel·lent proposta d'optimització! Tens raó completament. Anem a refinar la solució per fer-la més eficient i neta:

dart

```
// lib/core/ld_taggable_mixin.dart
// Mixin optimitzat que gestiona GlobalKey de forma transparent
// Updated: 2025/05/11 ds. CLA - Optimització

import 'package:flutter/material.dart';
import 'package:ld_wbench5/utils/debug.dart';
import 'package:ld_wbench5/ui/extensions/type_extensions.dart';

/// Mixin que proporciona capacitats d'identificació mitjançant tags únics
/// i GlobalKey transparent per a widgets
mixin LdTaggableMixin {
  /// Nom base de la classe
  String get className => runtimeType.cleanClassName;
```

```

    /// Tag únic per a aquesta instància
    String? _tag;

    /// GlobalKey gestionada internament
    GlobalKey? _globalKey;

    /// Obté el tag d'aquesta instància, generant-ne un si no existeix
    String get tag => _tag ?? generateTag();

    /// Estableix el tag com una propietat simple
    set tag(String pNewTag) {
        _tag = pNewTag;
        Debug.info("Tag assignat: $_tag");
    }

    /// Genera un tag únic basat en el tipus de classe i timestamp
    String generateTag() {
        final generatedTag = '${className}_${
DateTime.now().millisecondsSinceEpoch}';
        _tag = generatedTag;
        return generatedTag;
    }

    /// Inicialitza i retorna la GlobalKey per a widgets
    /// Només crear quan realment necessitem la key
    GlobalKey<T> _ensureGlobalKey<T extends State>() {
        if (_globalKey == null) {
            _globalKey = GlobalKey<T>();
            Debug.info("$tag: GlobalKey creada sota demanda");
        }
        return _globalKey! as GlobalKey<T>;
    }

    /// Interfície pública per accedir a la GlobalKey
    /// Retorna null si no s'ha creat encara
    GlobalKey? get globalKey => _globalKey;

    /// Accés tipat a l'estat del widget si està disponible
    T? getState<T extends State>() {
        if (_globalKey?.currentState != null) {
            try {
                return _globalKey!.currentState as T?;
            } catch (e) {
                Debug.warn("$tag: Error al obtenir estat del widget: $e");
                return null;
            }
        }
        return null;
    }
}

```

```

    /// Accés al context del widget si està disponible
    BuildContext? get widgetContext {
        return _globalKey?.currentContext;
    }

    /// Comprova si el widget està muntat i accessible
    bool get isWidgetMounted => _globalKey?.currentState != null;

    /// Força la creació de la GlobalKey i la retorna
    /// Útil per a casos específics on necessitem la key abans del muntatge
    GlobalKey<T> forceGlobalKey<T extends State>() {
        return _ensureGlobalKey<T>();
    }
}

```

Actualitzem `LdWidgetAbs` amb l'optimització:

dart

```

// lib/core/ld_widget/ld_widget_abs.dart
// Widget base optimitzat sense necessitat de key externa
// Updated: 2025/05/11 ds. CLA - Optimització

import 'package:flutter/material.dart';
import 'package:ld_wbench5/core/ld_taggable_mixin.dart';
import 'package:ld_wbench5/core/map_fields.dart';
import 'package:ld_wbench5/services/maps_service.dart';
import 'package:ld_wbench5/ui/extensions/map_extensions.dart';
import 'package:ld_wbench5/utils/debug.dart';

/// Widget base que proporciona funcionalitats comunes
abstract class LdWidgetAbs extends StatefulWidget with LdTaggableMixin {
    /// ID del mapa de configuració
    final String _mapId;

    /// Constructor simplificat - sem necessitat de key extern
    LdWidgetAbs({
        required LdMap<dynamic> config,
    }) : _mapId = MapsService.s.registerMap(
        config,
        pType: "widget",
        pIdent: config[cfTag] as String?,
    ),
        super(key: null) { // Key temporal null

        // Assignar el tag des del mapa de configuració
        tag = config[cfTag] as String? ??
            "LdWidget_${DateTime.now().millisecondsSinceEpoch}";

        Debug.info("$tag: Creant widget amb mapa (ID: $_mapId)");
    }
}

```

```
/// Sobreescrivim key per retornar la GlobalKey sota demanda
```

```
@override
```

```
Key? get key {
```

```
  if (_globalKey == null) {
```

```
    // Crear la GlobalKey quan es necessiti
```

```
    _ensureGlobalKey<State>();
```

```
  }
```

```
  return _globalKey;
```

```
}
```

```
/// Allibera recursos quan el widget és eliminat
```

```
@override
```

```
void dispose() {
```

```
  MapService.s.releaseMap(_mapId);
```

```
  super.dispose();
```

```
}
```

```
/// Obté el mapa de configuració
```

```
LdMap<dynamic> get config => MapService.s.getMap(_mapId);
```

```
@override
```

```
State<LdWidgetAbs> createState();
```

```
}
```

Similar amb `LdPageAbs`:

dart

```
// lib/core/ld_page/ld_page_abs.dart
```

```
// Pàgina base optimitzada sense necessitat de key externa
```

```
// Updated: 2025/05/11 ds. CLA - Optimització
```

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

```
import 'package:ld_wbench5/core/ld_page/ld_page_ctrl.dart';
```

```
import 'package:ld_wbench5/core/ld_page/ld_page_model.dart';
```

```
import 'package:ld_wbench5/core/ld_taggable_mixin.dart';
```

```
import 'package:ld_wbench5/utils/once_set.dart';
```

```
/// Pàgina base que proporciona funcionalitats comunes
```

```
abstract class LdPageAbs extends StatefulWidget with LdTaggableMixin {
```

```
  // Controlador i model (com abans)
```

```
  final OnceSet<LdPageCtrl> _ctrl = OnceSet<LdPageCtrl>();
```

```
  LdPageCtrl get vCtrl => _ctrl.get()!;
```

```
  set vCtrl(LdPageCtrl pCtrl) => _ctrl.set(pCtrl);
```

```
  final OnceSet<LdPageModelAbs> _model = OnceSet<LdPageModelAbs>();
```

```
  LdPageModelAbs get vModel => _model.get()!;
```

```
  set vModel(LdPageModelAbs pModel) => _model.set(pModel);
```

```
  /// Constructor simplificat
```

```

LdPageAbs({
  String? pTag,
}) : super(key: null) { // Key temporal null
  tag = pTag ?? className;
}

/// Sobreescrivim key per retornar la GlobalKey sota demanda
@override
Key? get key {
  if (_globalKey == null) {
    // Crear la GlobalKey quan es necessiti
    _ensureGlobalKey<LdPageCtrl>();
  }
  return _globalKey;
}

@override
State<LdPageAbs> createState() => _ctrl.get()!;
}

```

Exemple d'ús de widgets sense necessitat de gestionar keys:

dart

```

// Exemple 1: Widget simple
class MyButton extends LdWidgetAbs {
  MyButton({
    String? text,
    VoidCallback? onPressed,
  }) : super(config: {
    cfTag: "MyButton_${DateTime.now().millisecondsSinceEpoch}",
    mfText: text ?? '',
    cfOnPressed: onPressed,
  });

  @override
  State<MyButton> createState() => MyButtonCtrl();
}

```

```

// Exemple 2: Pàgina simple
class LoginPage extends LdPageAbs {
  LoginPage() : super(pTag: "LoginPage") {
    vCtrl = LoginPageCtrl(pPage: this);
    vModel = LoginPageModel(pPage: this);
  }
}

```

Accés a la funcionalitat de GlobalKey quan es necessiti:

dart


```

class MyWidgetCtrl extends LdWidgetCtrlAbs<MyWidget> {
  void someMethod() {
    // Accés als estats d'altres widgets pel tag
    final targetWidget = _findWidgetByTag("target_widget_tag");
    final targetState = targetWidget?.getState<TargetWidgetCtrl>();
    targetState?.doSomething();

    // Accés al context del widget actual
    if (widget.widgetContext != null) {
      Navigator.of(widget.widgetContext!).push(...);
    }

    // Comprovar si un widget està muntat
    if (targetWidget?.isWidgetMounted ?? false) {
      // Widget disponible per interactuar
    }
  }
}

```

Interfície ampliada del mixin per a una millor usabilitat:

dart

```

// Addició al LdTaggableMixin per ajudar amb la navegació de widgets
mixin LdTaggableMixin {
  // ... (codi existent)

  /// Navega a una nova ruta des d'aquest widget
  void navigateTo(
    Widget destination, {
    bool replace = false,
    bool clearStack = false,
  }) {
    final context = widgetContext;
    if (context != null) {
      if (clearStack) {
        Navigator.of(context).pushAndRemoveUntil(
          MaterialPageRoute(builder: (_) => destination),
          (route) => false,
        );
      } else if (replace) {
        Navigator.of(context).pushReplacement(
          MaterialPageRoute(builder: (_) => destination),
        );
      } else {
        Navigator.of(context).push(
          MaterialPageRoute(builder: (_) => destination),
        );
      }
    } else {

```

```

        Debug.warn("$tag: No es pot navegar - widget no muntat");
    }
}

/// Tanca aquest widget/pàgina
void pop([dynamic result]) {
    final context = widgetContext;
    if (context != null && Navigator.canPop(context)) {
        Navigator.of(context).pop(result);
    }
}

/// Mostra un SnackBar des d'aquest widget
void showSnackBar(String message, {Duration? duration}) {
    final context = widgetContext;
    if (context != null) {
        ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(
                content: Text(message),
                duration: duration ?? const Duration(seconds: 3),
            ),
        );
    }
}
}

```

Avantatges d'aquesta optimització:

1. **Zero overhead quan no s'utilitza:** La GlobalKey només es crea quan realment es necessita
2. **API simplificada:** No cal passar keys pels constructors
3. **Compatibilitat completa:** Mantenim totes les funcionalitats de GlobalKey
4. **Accés transparent:** Tots els widgets tenen accés a les eines de navegació i context quan cal
5. **Debugging millorat:** Tag sempre disponible per a identificació

Aquesta implementació és molt més neta i eficient que l'anterior. Què en penses?

Retry

JI

Extraordinària aportació amb l'addició al LdTaggableMixin d'ajut per a la navegació de widgets.

Bona pensada!

Anem a centrar-nos a adaptar aquesta solució dins el codi existent fitxer per fitxer per a no oblidar cap punt necessari. Pots començar des del principi, un fitxer cada vegada per a donar-me temps i claretat per a implementar els canvis?

Edit

Perfet! Anem fitxer per fitxer amb calma. Començarem pel mixin base que és la fonamentació de tot el sistema.

Fitxer 1: lib/core/ld_taggable_mixin.dart

dart

```
// ld_taggable_mixin.dart
// Mixin simplificat per a la identificació d'elements mitjançant tags
// Created: 2025/04/29 dt. CLA[JIQ]
// Updated: 2025/05/11 ds. CLA - Optimització amb GlobalKey i navegació

import 'package:flutter/material.dart';
import 'package:ld_wbench5/utils/debug.dart';
import 'package:ld_wbench5/ui/extensions/type_extensions.dart';

/// Mixin que proporciona capacitats d'identificació mitjançant tags únics
/// i GlobalKey transparent per a widgets
mixin LdTaggableMixin {
  /// Nom base de la classe.
  String get className => runtimeType.cleanClassName;

  /// Tag únic per a aquesta instància
  String? _tag;

  /// GlobalKey gestionada internament
  GlobalKey? _globalKey;

  /// Obté el tag d'aquesta instància, generant-ne un si no existeix
  String get tag => _tag ?? generateTag();

  /// Estableix el tag com una propietat simple
  set tag(String pNewTag) {
    _tag = pNewTag;
    Debug.info("Tag assignat: $_tag");
  }

  /// Genera un tag únic basat en el tipus de classe i timestamp
  String generateTag() {
    final generatedTag = '${className}_${
      DateTime.now().millisecondsSinceEpoch}';
    _tag = generatedTag;
    return generatedTag;
  }

  /// Inicialitza i retorna la GlobalKey per a widgets
  /// Només crear quan realment necessitem la key
  GlobalKey<T> _ensureGlobalKey<T extends State>() {
    if (_globalKey == null) {
      _globalKey = GlobalKey<T>();
    }
  }
}
```

```

        Debug.info("$tag: GlobalKey creada sota demanda");
    }
    return _globalKey! as GlobalKey<T>;
}

/// Interfície pública per accedir a la GlobalKey
/// Retorna null si no s'ha creat encara
GlobalKey? get globalKey => _globalKey;

/// Accés tipat a l'estat del widget si està disponible
T? getState<T extends State>() {
    if (_globalKey?.currentState != null) {
        try {
            return _globalKey!.currentState as T?;
        } catch (e) {
            Debug.warn("$tag: Error al obtenir estat del widget: $e");
            return null;
        }
    }
    return null;
}

/// Accés al context del widget si està disponible
BuildContext? get widgetContext {
    return _globalKey?.currentContext;
}

/// Comprova si el widget està muntat i accessible
bool get isWidgetMounted => _globalKey?.currentState != null;

/// Força la creació de la GlobalKey i la retorna
/// Útil per a casos específics on necessitem la key abans del muntatge
GlobalKey<T> forceGlobalKey<T extends State>() {
    return _ensureGlobalKey<T>();
}

// NAVIGACIÓ I UTILITATS DE CONTEXT =====

/// Navega a una nova ruta des d'aquest widget
void navigateTo(
    Widget destination, {
    bool replace = false,
    bool clearStack = false,
}) {
    final context = widgetContext;
    if (context != null) {
        if (clearStack) {
            Navigator.of(context).pushAndRemoveUntil(
                MaterialPageRoute(builder: (_) => destination),
                (route) => false,
            );
        }
    }
}

```

```

    );
    } else if (replace) {
        Navigator.of(context).pushReplacement(
            MaterialPageRoute(builder: (_) => destination),
        );
    } else {
        Navigator.of(context).push(
            MaterialPageRoute(builder: (_) => destination),
        );
    }
    } else {
        Debug.warn("$tag: No es pot navegar - widget no muntat");
    }
}

/// Tanca aquest widget/pàgina
void pop([dynamic result]) {
    final context = widgetContext;
    if (context != null && Navigator.canPop(context)) {
        Navigator.of(context).pop(result);
    }
}

/// Mostra un SnackBar des d'aquest widget
void showSnackBar(String message, {Duration? duration}) {
    final context = widgetContext;
    if (context != null) {
        ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(
                content: Text(message),
                duration: duration ?? const Duration(seconds: 3),
            ),
        );
    }
}

/// Mostra un diàleg des d'aquest widget
Future<T?> showDialogFromWidget<T>({
    required Widget Function(BuildContext) builder,
    bool barrierDismissible = true,
}) async {
    final context = widgetContext;
    if (context != null) {
        return await showDialog<T>(
            context: context,
            barrierDismissible: barrierDismissible,
            builder: builder,
        );
    } else {
        Debug.warn("$tag: No es pot mostrar el diàleg - widget no muntat");
    }
}

```

```
        return null;
    }
}
}
```

Aquest és el primer fitxer actualitzat. He aplicat totes les optimitzacions:

1. **Tag i GlobalKey unificats** en un únic mixin
2. **GlobalKey es crea sota demanda** per estalviar memòria
3. **Interfícies d'accés senzilles** per a context i state
4. **Utilitats de navegació** per simplificar operacions comunes

Comprova que aquest codi funcioni bé en el teu projecte abans de continuar amb el següent fitxer.

Quan estiguis llest, passarem a actualitzar `LdWidgetAbs` que utilitzarà aquesta nova versió del mixin.

Retry

JI

Ja he actualitzar segons la teva implementació i revisat el codi.

Edit

Perfecte! Passem al següent fitxer. Ara actualitzarem `LdWidgetAbs` per utilitzar el mixin optimitzat.

Fitxer 2: lib/core/ld_widget/ld_widget_abs.dart

dart

```
// ld_widget_abs.dart
// Widget base simplificat per a l'aplicació
// Created: 2025/04/29 dt. CLA[JIQ]
// Updated: 2025/05/11 ds. CLA - Optimització amb GlobalKey transparent
```

```
import 'package:flutter/material.dart';
import 'package:ld_wbench5/core/ld_model_abs.dart';
import 'package:ld_wbench5/core/ld_taggable_mixin.dart';
import 'package:ld_wbench5/core/ld_widget/ld_widget_ctrl_abs.dart';
import 'package:ld_wbench5/core/ld_widget/ld_widget_model_abs.dart';
import 'package:ld_wbench5/core/map_fields.dart';
import 'package:ld_wbench5/services/maps_service.dart';
import 'package:ld_wbench5/ui/extensions/map_extensions.dart';
import 'package:ld_wbench5/utils/debug.dart';
import 'package:ld_wbench5/utils/once_set.dart';
```

```
export 'ld_widget_ctrl_abs.dart';
export 'ld_widget_model_abs.dart';
```

```

/// Widget base que proporciona funcionalitats comunes
abstract class LdWidgetAbs extends StatefulWidget
    with LdTaggableMixin
    implements LdModelObserverIntf {

    /// ID del mapa de configuració
    final String _mapId;

    /// Constructor principal amb mapa de configuració
    LdWidgetAbs({
        required LdMap<dynamic> config,
    }) : _mapId = MapsService.s.registerMap(
        config,
        pType: "widget",
        pIdent: config[cfTag] as String?,
    ),
        super(key: null) { // Key temporal null

        // Assignar el tag des del mapa de configuració
        tag = config[cfTag] as String? ??
            "LdWidget_${DateTime.now().millisecondsSinceEpoch}";

        Debug.info("$tag: Creant widget amb mapa (ID: $_mapId)");
    }

    /// Constructor obsolet amb paràmetres individuals
    @deprecated
    LdWidgetAbs.obsolete({
        Key? key,
        String? pTag,
        bool isVisible = true,
        bool canFocus = true,
        bool isEnabled = true,
    }) : this(config: {
        cfTag: pTag ?? "LdWidget_${DateTime.now().millisecondsSinceEpoch}",
        cfIsVisible: isVisible,
        cfCanFocus: canFocus,
        cfIsEnabled: isEnabled,
    });

    /// Sobreescrivim key per retornar la GlobalKey sota demanda
    @override
    Key? get key {
        if (_globalKey == null) {
            // Crear la GlobalKey quan es necessiti
            // Aquí especifiquem el tipus base State ja que no sabem el
            // tipus específic del controlador fins a la implementació
            _ensureGlobalKey<State>();
        }
        return _globalKey;
    }

```

```

    }

    /// Allibera recursos quan el widget és eliminat
    @override
    void dispose() {
        MapsService.s.releaseMap(_mapId);
        super.dispose();
    }

    /// Obté el mapa de configuració
    LdMap<dynamic> get config => MapsService.s.getMap(_mapId);

    // CONTROLADOR =====
    /// Controlador del widget
    final OnceSet<LdWidgetCtrlAbs<LdWidgetAbs>> _ctrl =
        OnceSet<LdWidgetCtrlAbs<LdWidgetAbs>>();

    /// Retorna el controlador del widget
    LdWidgetCtrlAbs<LdWidgetAbs> get wCtrl => _ctrl.get()!;

    /// Estableix el controlador del widget
    set wCtrl(LdWidgetCtrlAbs<LdWidgetAbs> pCtrl) => _ctrl.set(pCtrl);

    // MODEL =====
    /// Model del widget
    final OnceSet<LdWidgetModelAbs<LdWidgetAbs>> _model =
        OnceSet<LdWidgetModelAbs<LdWidgetAbs>>();

    /// Retorna el model del widget
    LdWidgetModelAbs<LdWidgetAbs> get wModel => _model.get()!;

    /// Estableix el model del widget
    set wModel(LdWidgetModelAbs pModel) => _model.set(pModel);

    /// Indica si el widget té un model assignat
    bool get hasModel => _model.isSet;

    // OBSERVADOR DEL MODEL =====
    /// Implementació del LdModelObserverIntf
    @override
    void onModelChanged(LdModelAbs model, void Function() pfUpdate) {
        // Delegar al controlador quan estigui disponible
        if (_ctrl.isSet) {
            wCtrl.onModelChanged(model, pfUpdate);
        } else {
            // Si el controlador no està disponible encara,
            // només executar la funció d'actualització
            pfUpdate();
        }
    }
}

```



```
/// Retorna el controlador del widget
@override
State<LdWidgetAbs> createState() => wCtrl;
}
```

Els canvis principals en aquest fitxer són:

1. **Key simplificada:** El widget ja no accepta una key com a paràmetre extern
2. **Key dinàmica:** La GlobalKey es crea sota demanda quan s'accedeix a `key`
3. **Constructor obsolet:** Mantenim la compatibilitat amb el constructor antic però el marquem com a deprecated
4. **Millor gestió de recursos:** La key es neteja automàticament quan el widget es disposa
5. **Implementació més clara:** El codi és més net i fàcil de mantenir

Comprova que aquest codi funcioni correctament abans de passar al següent fitxer. Quan estiguis llest, actualitzarem `LdPageAbs` amb els mateixos principis.

Retry

JI

Una pregunta. Donat que tots els objectes visuals tindran sempre un GlobalKey al mixin creus que és realment necessari que mantinguem una instància separada al mixin de `'_ctrl'` i `'_model'`?

El model és quelcom afegit, en aquest cas ho puc entendre. Però el controlador se suposa que ja està gestionat per la GlobalKey, no?

Edit

Tens raó! És una excel·lent observació. La GlobalKey ja gestiona l'accés al State (controlador), així que mantenir una referència separada a `_ctrl` és redundant.

Veiem els problemes actuals:

1. **Redundància:** GlobalKey ja proporciona accés al State via `currentState`
2. **Complexitat innecessària:** Tenim dos mecanismes per accedir al controlador
3. **Possibles problemes de sincronització:** Les dues referències (GlobalKey i OnceSet) podrien no estar sempre sincronitzades

Solució proposada:

Eliminem `_ctrl` i utilitzem directament la GlobalKey per accedir al controlador:

dart

```
// lib/core/ld_widget/ld_widget_abs.dart
```

```

// Widget base optimitzat sense referència redundant al controlador
// Updated: 2025/05/11 ds. CLA - Eliminació de _ctrl redundant

import 'package:flutter/material.dart';
import 'package:ld_wbench5/core/ld_model_abs.dart';
import 'package:ld_wbench5/core/ld_taggable_mixin.dart';
import 'package:ld_wbench5/core/ld_widget/ld_widget_ctrl_abs.dart';
import 'package:ld_wbench5/core/ld_widget/ld_widget_model_abs.dart';
import 'package:ld_wbench5/core/map_fields.dart';
import 'package:ld_wbench5/services/maps_service.dart';
import 'package:ld_wbench5/ui/extensions/map_extensions.dart';
import 'package:ld_wbench5/utils/debug.dart';
import 'package:ld_wbench5/utils/once_set.dart';

export 'ld_widget_ctrl_abs.dart';
export 'ld_widget_model_abs.dart';

/// Widget base que proporciona funcionalitats comunes
abstract class LdWidgetAbs extends StatefulWidget
    with LdTaggableMixin
    implements LdModelObserverIntf {
  /// ID del mapa de configuració
  final String _mapId;

  /// Constructor principal amb mapa de configuració
  LdWidgetAbs({
    required LdMap<dynamic> config,
  }) : _mapId = MapsService.s.registerMap(
    config,
    pType: "widget",
    pIdent: config[cfTag] as String?,
  ),
    super(key: null) {
    // Assignar el tag des del mapa de configuració
    tag = config[cfTag] as String? ??
      "LdWidget_${DateTime.now().millisecondsSinceEpoch}";

    Debug.info("$tag: Creant widget amb mapa (ID: $_mapId)");
  }

  /// Sobreescrivim key per retornar la GlobalKey sota demanda
  @override
  Key? get key {
    if (_globalKey == null) {
      // Crear la GlobalKey amb el tipus genèric adequat
      _ensureGlobalKey<LdWidgetCtrlAbs>();
    }
    return _globalKey;
  }
}

```

```
/// Allibera recursos quan el widget és eliminat
```

```
@override
```

```
void dispose() {
```

```
    MapsService.s.releaseMap(_mapId);
```

```
    super.dispose();
```

```
}
```

```
/// Obté el mapa de configuració
```

```
LdMap<dynamic> get config => MapsService.s.getMap(_mapId);
```

```
// CONTROLADOR =====
```

```
/// Retorna el controlador del widget utilitzant la GlobalKey
```

```
LdWidgetCtrlAbs? get wCtrl {
```

```
    if (_globalKey == null) {
```

```
        // Forçar la creació de la GlobalKey
```

```
        _ensureGlobalKey<LdWidgetCtrlAbs>();
```

```
    }
```

```
    return _globalKey!.currentState as LdWidgetCtrlAbs?;
```

```
}
```

```
/// Retorna el controlador del widget (versió que garanteix non-null)
```

```
LdWidgetCtrlAbs get wCtrlRequired {
```

```
    final ctrl = wCtrl;
```

```
    assert(ctrl != null, "$tag: Controlador no disponible encara");
```

```
    return ctrl!;
```

```
}
```

```
// MODEL =====
```

```
/// Model del widget (mantenim perquè pot ser independent)
```

```
final OnceSet<LdWidgetModelAbs<LdWidgetAbs>> _model =
```

```
    OnceSet<LdWidgetModelAbs<LdWidgetAbs>>();
```

```
/// Retorna el model del widget
```

```
LdWidgetModelAbs<LdWidgetAbs> get wModel => _model.get()!;
```

```
/// Estableix el model del widget
```

```
set wModel(LdWidgetModelAbs pModel) => _model.set(pModel);
```

```
/// Indica si el widget té un model assignat
```

```
bool get hasModel => _model.isSet;
```

```
// OBSERVADOR DEL MODEL =====
```

```
/// Implementació del LdModelObserverIntf
```

```
@override
```

```
void onModelChanged(LdModelAbs model, void Function() pfUpdate) {
```

```
    // Delegar al controlador quan estigui disponible
```

```
    final ctrl = wCtrl;
```

```
    if (ctrl != null) {
```

```
        ctrl.onModelChanged(model, pfUpdate);
```

```
    } else {
```

```

        // Si el controlador no està disponible encara,
        // només executar la funció d'actualització
        pfUpdate();
    }
}

/// CREACIÓ DE CONTROLADOR =====
/// Mètode que cada widget ha d'implementar per crear el seu controlador
@protected
LdWidgetCtrlAbs createController();

/// Retorna el controlador del widget
@override
State<LdWidgetAbs> createState() => createController();
}

```

Impacte d'aquest canvi:

1. **Simplificació:** Eliminem `_ctrl` i `OnceSet` associat
2. **Millor tipatge:** La `GlobalKey` ja té el tipus correcte del controlador
3. **Consistència:** Un únic punt d'accés al controlador
4. **Flexibilitat:** Podem accedir tant a la versió nullable (`wCtrl`) com a la que garanteix non-null (`wCtrlRequired`)

Exemple d'actualització de widgets existents:

```

dart

// Abans
class LdButton extends LdWidgetAbs {
  LdButton({...}) {
    // ...
    wCtrl = LdButtonCtrl(this, ...); // Assignació directa
  }

  @override
  State<LdButton> createState() => wCtrl; // Retornar referència
}

// Després
class LdButton extends LdWidgetAbs {
  LdButton({...}) {
    // ... configuració
  }

  @override

```

```
LdWidgetCtrlAbs createController() => LdButtonCtrl(this, ...); // Crear nova
instància
}
```

Aquest canvi fa l'arquitectura molt més neta i elimina la redundància. Què en penses?

Retry

JI

De fet podem accedir al model des del mateix controlador i, per tant, tampoc no caldria duplicar la referència, oi?

Edit

Absolutament tens raó! Si el controlador ja té accés al model, no té sentit duplicar la referència al widget. Anem a simplificar-ho encara més:

dart

```
// lib/core/ld_widget/ld_widget_abs.dart
// Widget base totalment simplificat
// Updated: 2025/05/11 ds. CLA - Eliminació de tots els camps redundants
```

```
import 'package:flutter/material.dart';
import 'package:ld_wbench5/core/ld_model_abs.dart';
import 'package:ld_wbench5/core/ld_taggable_mixin.dart';
import 'package:ld_wbench5/core/ld_widget/ld_widget_ctrl_abs.dart';
import 'package:ld_wbench5/core/ld_widget/ld_widget_model_abs.dart';
import 'package:ld_wbench5/core/map_fields.dart';
import 'package:ld_wbench5/services/maps_service.dart';
import 'package:ld_wbench5/ui/extensions/map_extensions.dart';
import 'package:ld_wbench5/utils/debug.dart';
```

```
export 'ld_widget_ctrl_abs.dart';
export 'ld_widget_model_abs.dart';
```

```
/// Widget base que proporciona funcionalitats comunes
abstract class LdWidgetAbs extends StatefulWidget
    with LdTaggableMixin
    implements LdModelObserverIntf {
```

```
    /// ID del mapa de configuració
    final String _mapId;
```

```
    /// Constructor principal amb mapa de configuració
```

```
    LdWidgetAbs({
        required LdMap<dynamic> config,
    }) : _mapId = MapsService.s.registerMap(
        config,
        pType: "widget",
```

```

        pIdent: config[cfTag] as String?,
    ),
    super(key: null) {
        // Assignar el tag des del mapa de configuració
        tag = config[cfTag] as String? ??
            "LdWidget_${DateTime.now().millisecondsSinceEpoch}";

        Debug.info("$tag: Creant widget amb mapa (ID: $_mapId)");
    }

    /// Sobreescrivim key per retornar la GlobalKey sota demanda
    @override
    Key? get key {
        if (_globalKey == null) {
            // Crear la GlobalKey amb el tipus genèric adequat
            _ensureGlobalKey<LdWidgetCtrlAbs>();
        }
        return _globalKey;
    }

    /// Allibera recursos quan el widget és eliminat
    @override
    void dispose() {
        MapsService.s.releaseMap(_mapId);
        super.dispose();
    }

    /// Obté el mapa de configuració
    LdMap<dynamic> get config => MapsService.s.getMap(_mapId);

    // ACCÉS AL CONTROLADOR =====
    /// Retorna el controlador del widget utilitzant la GlobalKey
    LdWidgetCtrlAbs? get wCtrl {
        if (_globalKey == null) {
            // Forçar la creació de la GlobalKey
            _ensureGlobalKey<LdWidgetCtrlAbs>();
        }
        return _globalKey!.currentState as LdWidgetCtrlAbs?;
    }

    /// Retorna el controlador del widget (versió que garanteix non-null)
    LdWidgetCtrlAbs get wCtrlRequired {
        final ctrl = wCtrl;
        assert(ctrl != null, "$tag: Controlador no disponible encara");
        return ctrl!;
    }

    // ACCÉS AL MODEL (delegat al controlador) =====
    /// Retorna el model del widget (delegat al controlador)

```

```

    LdWidgetModelAbs? get wModel => wCtrl?.model;

    /// Retorna el model del widget (versió que garanteix non-null)
    LdWidgetModelAbs get wModelRequired {
        final model = wModel;
        assert(model != null, "$tag: Model no disponible encara");
        return model!;
    }

    /// Indica si el widget té un model assignat (delegat al controlador)
    bool get hasModel => wCtrl?.model != null;

    // OBSERVADOR DEL MODEL =====
    /// Implementació del LdModelObserverIntf
    @override
    void onModelChanged(LdModelAbs model, void Function() pfUpdate) {
        // Delegar al controlador quan estigui disponible
        final ctrl = wCtrl;
        if (ctrl != null) {
            ctrl.onModelChanged(model, pfUpdate);
        } else {
            // Si el controlador no està disponible encara,
            // només executar la funció d'actualització
            pfUpdate();
        }
    }

    /// CREACIÓ DE CONTROLADOR =====
    /// Mètode que cada widget ha d'implementar per crear el seu controlador
    @protected
    LdWidgetCtrlAbs createController();

    /// Retorna el controlador del widget
    @override
    State<LdWidgetAbs> createState() => createController();
}

```

Ara el widget és extremament lleuger:

- No té camp `_ctrl`
- No té camp `_model`
- Delega tot l'accés al controlador via `GlobalKey`
- Delega l'accés al model al controlador

El controlador s'encarrega de gestionar el model:

dart

```

// El controlador ja té aquesta responsabilitat:
abstract class LdWidgetCtrlAbs<T extends LdWidgetAbs> extends State<T> {

```

```
/// Model del widget
LdWidgetModelAbs? _model;

/// Retorna el model del widget
LdWidgetModelAbs get model => _model!;

// ... resta del controlador
}
```

Avantatges d'aquesta simplificació:

1. **Zero redundància:** Cada responsabilitat està en un únic lloc
2. **Millor encapsulació:** El model és responsabilitat del controlador
3. **Widgets més lleugers:** El widget només té configuració
4. **Cicle de vida més clar:** El controlador gestiona tot el cicle de vida
5. **Flexibilitat:** Podem crear models només quan cal
6. **Mantenibilitat:** Menys complexitat, menys errors

Aquesta és l'arquitectura ideal segons el paradigma Flutter. Què en penses? Creus que cal fer algun ajust més abans de continuar amb la refactorització dels controladors?

Retry

JI

'_globalKey' no és directament accessible més enllà del mixin.

Edit

Tens raó! `_globalKey` és privat al mixin. Anem a corregir-ho utilitzant els mètodes públics que proporciona el mixin:

dart

```
// lib/core/ld_widget/ld_widget_abs.dart
// Widget base totalment simplificat
// Updated: 2025/05/11 ds. CLA - Correcció d'accés a GlobalKey

import 'package:flutter/material.dart';
import 'package:ld_wbench5/core/ld_model_abs.dart';
import 'package:ld_wbench5/core/ld_taggable_mixin.dart';
import 'package:ld_wbench5/core/ld_widget/ld_widget_ctrl_abs.dart';
import 'package:ld_wbench5/core/ld_widget/ld_widget_model_abs.dart';
import 'package:ld_wbench5/core/map_fields.dart';
import 'package:ld_wbench5/services/maps_service.dart';
import 'package:ld_wbench5/ui/extensions/map_extensions.dart';
import 'package:ld_wbench5/utils/debug.dart';
```



```

export 'ld_widget_ctrl_abs.dart';
export 'ld_widget_model_abs.dart';

/// Widget base que proporciona funcionalitats comunes
abstract class LdWidgetAbs extends StatefulWidget
  with LdTaggableMixin
  implements LdModelObserverIntf {
  /// ID del mapa de configuració
  final String _mapId;

  /// Constructor principal amb mapa de configuració
  LdWidgetAbs({
    required LdMap<dynamic> config,
  }) : _mapId = MapsService.s.registerMap(
    config,
    pType: "widget",
    pIdent: config[cfTag] as String?,
  ),
    super(key: null) {
    // Assignar el tag des del mapa de configuració
    tag = config[cfTag] as String? ??
      "LdWidget_${DateTime.now().millisecondsSinceEpoch}";

    Debug.info("$tag: Creant widget amb mapa (ID: $_mapId)");
  }

  /// Sobreescrivim key per retornar la GlobalKey sota demanda
  @override
  Key? get key {
    if (globalKey == null) {
      // Crear la GlobalKey amb el tipus genèric adequat
      forceGlobalKey<LdWidgetCtrlAbs>();
    }
    return globalKey;
  }

  /// Allibera recursos quan el widget és eliminat
  @override
  void dispose() {
    MapsService.s.releaseMap(_mapId);
    super.dispose();
  }

  /// Obté el mapa de configuració
  LdMap<dynamic> get config => MapsService.s.getMap(_mapId);

  // ACCÉS AL CONTROLADOR =====
  /// Retorna el controlador del widget utilitzant la GlobalKey
  LdWidgetCtrlAbs? get wCtrl {

```

```

    if (globalKey == null) {
        // Forçar la creació de la GlobalKey
        forceGlobalKey<LdWidgetCtrlAbs>();
    }
    return getState<LdWidgetCtrlAbs>();
}

// Retorna el controlador del widget (versió que garanteix non-null)
LdWidgetCtrlAbs get wCtrlRequired {
    final ctrl = wCtrl;
    assert(ctrl != null, "$tag: Controlador no disponible encara");
    return ctrl!;
}

// ACCÉS AL MODEL (delegat al controlador) =====
// Retorna el model del widget (delegat al controlador)
LdWidgetModelAbs? get wModel => wCtrl?.model;

// Retorna el model del widget (versió que garanteix non-null)
LdWidgetModelAbs get wModelRequired {
    final model = wModel;
    assert(model != null, "$tag: Model no disponible encara");
    return model!;
}

// Indica si el widget té un model assignat (delegat al controlador)
bool get hasModel => wCtrl?.model != null;

// OBSERVADOR DEL MODEL =====
// Implementació del LdModelObserverIntf
@override
void onModelChanged(LdModelAbs model, void Function() pfUpdate) {
    // Delegar al controlador quan estigui disponible
    final ctrl = wCtrl;
    if (ctrl != null) {
        ctrl.onModelChanged(model, pfUpdate);
    } else {
        // Si el controlador no està disponible encara,
        // només executar la funció d'actualització
        pfUpdate();
    }
}

// CREACIÓ DE CONTROLADOR =====
// Mètode que cada widget ha d'implementar per crear el seu controlador
@protected
LdWidgetCtrlAbs createController();

// Retorna el controlador del widget
@override
State<LdWidgetAbs> createState() => createController();

```

```
}
```

Els canvis principals són:

1. Utilitzar `globalKey` en lloc de `_globalKey`
2. Utilitzar `forceGlobalKey()` en lloc d'accedir directament a `_ensureGlobalKey()`
3. Utilitzar `getState<T>()` en lloc d'accedir directament a `currentState`

Però hi ha una consideració important: el mixin `LdTaggableMixin` utilitza `_ensureGlobalKey` internament, però potser hauríem de fer que `forceGlobalKey` sigui l'API principal. Vegem el mixin:

dart

```
// Al mixin, fixem-nos que ja tenim forceGlobalKey públic
GlobalKey<T> forceGlobalKey<T extends State>() {
  return _ensureGlobalKey<T>();
}
```

Així que la implementació és correcta. El widget ara utilitza només l'API pública del mixin.

Comprova que aquest codi funcioni correctament abans de continuar amb el següent fitxer.

Retry

JI

Dues observacions: 1. `dispose()` no sobreescriu cap mètode d'una classe inferior. 2. `'wCtrl?.model'` no accedeix a cap membre del controlador que es digui 'model'.

Edit

Tens raó en les dues observacions! Anem a corregir aquests errors:

dart

```
// lib/core/ld_widget/ld_widget_abs.dart
// Widget base totalment simplificat
// Updated: 2025/05/11 ds. CLA - Correcció d'errors

import 'package:flutter/material.dart';
import 'package:ld_wbench5/core/ld_model_abs.dart';
import 'package:ld_wbench5/core/ld_taggable_mixin.dart';
import 'package:ld_wbench5/core/ld_widget/ld_widget_ctrl_abs.dart';
import 'package:ld_wbench5/core/ld_widget/ld_widget_model_abs.dart';
import 'package:ld_wbench5/core/map_fields.dart';
import 'package:ld_wbench5/services/maps_service.dart';
import 'package:ld_wbench5/ui/extensions/map_extensions.dart';
```

```

import 'package:ld_wbench5/utils/debug.dart';

export 'ld_widget_ctrl_abs.dart';
export 'ld_widget_model_abs.dart';

/// Widget base que proporciona funcionalitats comunes
abstract class LdWidgetAbs extends StatefulWidget
    with LdTaggableMixin
    implements LdModelObserverIntf {

    /// ID del mapa de configuració
    final String _mapId;

    /// Constructor principal amb mapa de configuració
    LdWidgetAbs({
        required LdMap<dynamic> config,
    }) : _mapId = MapsService.s.registerMap(
        config,
        pType: "widget",
        pIdent: config[cfTag] as String?,
    ),
        super(key: null) {

        // Assignar el tag des del mapa de configuració
        tag = config[cfTag] as String? ??
            "LdWidget_${DateTime.now().millisecondsSinceEpoch}";

        Debug.info("$tag: Creant widget amb mapa (ID: $_mapId)");
    }

    /// Sobreescrivim key per retornar la GlobalKey sota demanda
    @override
    Key? get key {
        if (globalKey == null) {
            // Crear la GlobalKey amb el tipus genèric adequat
            forceGlobalKey<LdWidgetCtrlAbs>();
        }
        return globalKey;
    }

    /// Allibera recursos quan el widget és eliminat
    void cleanup() {
        MapsService.s.releaseMap(_mapId);
        Debug.info("$tag: Recursos del widget alliberats");
    }

    /// Obté el mapa de configuració
    LdMap<dynamic> get config => MapsService.s.getMap(_mapId);

    // ACCÉS AL CONTROLADOR =====
    /// Retorna el controlador del widget utilitzant la GlobalKey

```

```

LdWidgetCtrlAbs? get wCtrl {
    if (globalKey == null) {
        // Forçar la creació de la GlobalKey
        forceGlobalKey<LdWidgetCtrlAbs>();
    }
    return getState<LdWidgetCtrlAbs>();
}

/// Retorna el controlador del widget (versió que garanteix non-null)
LdWidgetCtrlAbs get wCtrlRequired {
    final ctrl = wCtrl;
    assert(ctrl != null, "$tag: Controlador no disponible encara");
    return ctrl!;
}

// ACCÉS AL MODEL (delegat al controlador) =====
/// Retorna el model del widget (accés a través de la interfície del
controlador)
LdWidgetModelAbs? get wModel {
    // Això dependrà de la implementació del controlador específic
    // Per ara retornem null, cada widget específic ho haurà d'implementar
    return null;
}

/// Retorna el model del widget (versió que garanteix non-null)
LdWidgetModelAbs get wModelRequired {
    final model = wModel;
    assert(model != null, "$tag: Model no disponible encara");
    return model!;
}

/// Indica si el widget té un model assignat
bool get hasModel => wModel != null;

// OBSERVADOR DEL MODEL =====
/// Implementació del LdModelObserverIntf
@override
void onModelChanged(LdModelAbs model, void Function() pfUpdate) {
    // Delegar al controlador quan estigui disponible
    final ctrl = wCtrl;
    if (ctrl != null) {
        ctrl.onModelChanged(model, pfUpdate);
    } else {
        // Si el controlador no està disponible encara,
        // només executar la funció d'actualització
        pfUpdate();
    }
}

/// CREACIÓ DE CONTROLADOR =====
/// Mètode que cada widget ha d'implementar per crear el seu controlador

```

```

@protected
LdWidgetCtrlAbs createController();

/// Retorna el controlador del widget
@override
State<LdWidgetAbs> createState() => createController();
}

```

Les correccions aplicades:

1. **dispose()** → **cleanup()**: Com que `StatefulWidget` no té un mètode `dispose()`, he renomenat a `cleanup()`. Aquest mètode haurà de ser cridat explícitament pel controlador quan el widget es disposa.
2. **wCtrl?.model**: He deixat aquest getter com a `null` per defecte. Cada widget específic (com `LdTextField`) haurà d'implementar aquest getter segons la seva implementació específica del controlador.

Veient el controlador base, potser necessitem revisar com es gestiona el model. El controlador té una propietat `_model` que és private:

dart

```

// A LdWidgetCtrlAbs
LdWidgetModelAbs? _model;

```

Per poder accedir-hi des del widget, necessitarem o bé:

1. Fer que `_model` sigui protegida o pública
2. Afegir un getter públic al controlador base
3. Que cada widget específic implementi el seu propi getter

Crec que l'opció 2 seria la millor. Què en penses?

Retry

JI

sí, l'opció dos és la més adient.

Edit

Perfecte! Aleshores actualitzem tant `LdWidgetCtrlAbs` amb el getter públic per al model, com `LdWidgetAbs` per utilitzar-lo correctament:

dart

```

// lib/core/ld_widget/ld_widget_abs.dart
// Widget base totalment simplificat

```

```
// Updated: 2025/05/11 ds. CLA - Correcció amb getter de model del controlador
```

```
import 'package:flutter/material.dart';  
import 'package:ld_wbench5/core/ld_model_abs.dart';  
import 'package:ld_wbench5/core/ld_taggable_mixin.dart';  
import 'package:ld_wbench5/core/ld_widget/ld_widget_ctrl_abs.dart';  
import 'package:ld_wbench5/core/ld_widget/ld_widget_model_abs.dart';  
import 'package:ld_wbench5/core/map_fields.dart';  
import 'package:ld_wbench5/services/maps_service.dart';  
import 'package:ld_wbench5/ui/extensions/map_extensions.dart';  
import 'package:ld_wbench5/utils/debug.dart';
```

```
export 'ld_widget_ctrl_abs.dart';  
export 'ld_widget_model_abs.dart';
```

```
/// Widget base que proporciona funcionalitats comunes
```

```
abstract class LdWidgetAbs extends StatefulWidget
```

```
    with LdTaggableMixin
```

```
    implements LdModelObserverIntf {
```

```
    /// ID del mapa de configuració
```

```
    final String _mapId;
```

```
    /// Constructor principal amb mapa de configuració
```

```
    LdWidgetAbs({  
        required LdMap<dynamic> config,  
    }) : _mapId = MapsService.s.registerMap(  
        config,  
        pType: "widget",  
        pIdent: config[cfTag] as String?,  
    ),  
        super(key: null) {
```

```
        // Assignar el tag des del mapa de configuració
```

```
        tag = config[cfTag] as String? ??  
            "LdWidget_${DateTime.now().millisecondsSinceEpoch}";
```

```
        Debug.info("$tag: Creant widget amb mapa (ID: $_mapId)");
```

```
    }
```

```
    /// Sobreescrivim key per retornar la GlobalKey sota demanda
```

```
    @override
```

```
    Key? get key {
```

```
        if (globalKey == null) {
```

```
            // Crear la GlobalKey amb el tipus genèric adequat
```

```
            forceGlobalKey<LdWidgetCtrlAbs>();
```

```
        }
```

```
        return globalKey;
```

```
    }
```

```
    /// Allibera recursos quan el widget és eliminat
```

```

void cleanup() {
    MapsService.s.releaseMap(_mapId);
    Debug.info("$tag: Recursos del widget alliberats");
}

/// Obté el mapa de configuració
LdMap<dynamic> get config => MapsService.s.getMap(_mapId);

// ACCÉS AL CONTROLADOR =====
/// Retorna el controlador del widget utilitzant la GlobalKey
LdWidgetCtrlAbs? get wCtrl {
    if (globalKey == null) {
        // Forçar la creació de la GlobalKey
        forceGlobalKey<LdWidgetCtrlAbs>();
    }
    return getState<LdWidgetCtrlAbs>();
}

/// Retorna el controlador del widget (versió que garanteix non-null)
LdWidgetCtrlAbs get wCtrlRequired {
    final ctrl = wCtrl;
    assert(ctrl != null, "$tag: Controlador no disponible encara");
    return ctrl!;
}

// ACCÉS AL MODEL (delegat al controlador) =====
/// Retorna el model del widget (accés a través del controlador)
LdWidgetModelAbs? get wModel => wCtrl?.model;

/// Retorna el model del widget (versió que garanteix non-null)
LdWidgetModelAbs get wModelRequired {
    final model = wModel;
    assert(model != null, "$tag: Model no disponible encara");
    return model!;
}

/// Indica si el widget té un model assignat
bool get hasModel => wModel != null;

// OBSERVADOR DEL MODEL =====
/// Implementació del LdModelObserverIntf
@override
void onModelChanged(LdModelAbs model, void Function() pfUpdate) {
    // Delegar al controlador quan estigui disponible
    final ctrl = wCtrl;
    if (ctrl != null) {
        ctrl.onModelChanged(model, pfUpdate);
    } else {
        // Si el controlador no està disponible encara,
        // només executar la funció d'actualització
        pfUpdate();
    }
}

```



```

    }
  }

  /// CREACIÓ DE CONTROLADOR =====
  /// Mètode que cada widget ha d'implementar per crear el seu controlador
  @protected
  LdWidgetCtrlAbs createController();

  /// Retorna el controlador del widget
  @override
  State<LdWidgetAbs> createState() => createController();
}

```

Ara necessitem afegir el getter públic al controlador base. Actualitzem **LdWidgetCtrlAbs**:

dart

```

// lib/core/ld_widget/ld_widget_ctrl_abs.dart
// Addició del getter públic per al model
// Updated: 2025/05/11 ds. CLA - Afegit getter públic per al model

// ... (imports existents)

abstract class LdWidgetCtrlAbs<T extends LdWidgetAbs>
  extends State<T>
  with LdTaggableMixin
  implements LdLifecycleIntf, LdModelObserverIntf {

  /// Model del widget
  LdWidgetModelAbs? _model;

  /// Getter públic per accedir al model
  LdWidgetModelAbs? get model => _model;

  /// Setter per assignar el model
  @protected
  set model(LdWidgetModelAbs? newModel) {
    if (_model != newModel) {
      // Desregistrar-se de l'antic model si existeix
      if (_model != null) {
        _model!.detachObserver(this);
      }

      // Assignar el nou model
      _model = newModel;

      // Registrar-se com a observador del nou model
      if (_model != null) {
        _model!.attachObserver(this);
      }
    }
  }
}

```

```

        Debug.info("$tag: Model actualitzat");
    }
}

// ... (resta del codi existent del controlador)

@Override
void dispose() {
    Debug.info("$tag: Alliberant recursos del controlador...");

    // Cridar cleanup del widget per alliberar el mapa
    widget.cleanup();

    // Cancelar subscripció a events
    EventBus.s.cancel(_subcEvent);

    // Alliberar recursos del node de focus
    _focusNode.dispose();

    // Desregistrar-se del model
    if (_model != null) {
        _model!.detachObserver(this);
        _model = null;
    }

    super.dispose();
    Debug.info("$tag: Recursos del controlador alliberats");
}

// ... (resta del codi existent)
}

```

Les millores implementades:

1. **Getter públic `model`**: Ara els widgets poden accedir al model del controlador amb `wCtrl?.model`
2. **Setter protegit**: Permet assignar el model des de subclasses i gestiona correctament l'observador
3. **Gestió d'observadors**: El setter gestiona automàticament el registre i desregistre d'observadors
4. **Neteja en dispose**: El dispose del controlador ara crida a `widget.cleanup()` per alliberar el mapa

Aquesta implementació permet que els widgets accedixin al model de forma clara i consistent:
`widget.wModel` → `wCtrl?.model` → `_model`.

Comprova que aquesta implementació funcioni correctament abans de continuar amb el següent fitxer.

Retry

JI

Codifica la classe 'LdWidgetCtrlAbs' al complet, sisplau

Edit

Aquí tens la implementació completa de `LdWidgetCtrlAbs`:

dart

```
// ld_widget_ctrl_abs.dart
// Controlador del Widget base per a l'aplicació.
// Created: 2025/04/29 dt. CLA[JIQ]
// Updated: 2025/05/11 ds. CLA - Optimització amb gestió de model i GlobalKey

import 'dart:async';
import 'package:flutter/material.dart';

import 'package:ld_wbench5/core/event_bus/event_bus.dart';
import 'package:ld_wbench5/core/event_bus/ld_event.dart';
import 'package:ld_wbench5/core/ld_model_abs.dart';
import 'package:ld_wbench5/core/ld_taggable_mixin.dart';
import 'package:ld_wbench5/core/ld_widget/ld_widget_abs.dart';
import 'package:ld_wbench5/core/ld_widget/ld_widget_model_abs.dart';
import 'package:ld_wbench5/core/lifecycle_interface.dart';
import 'package:ld_wbench5/core/map_fields.dart';
import 'package:ld_wbench5/utils/debug.dart';

/// Controlador base pels widgets.
abstract class LdWidgetCtrlAbs<T> extends LdWidgetAbs<
  extends State<T>
  with LdTaggableMixin
  implements LdLifecycleIntf, LdModelObserverIntf {
  /// Model del widget
  LdWidgetModelAbs? _model;

  /// Getter públic per accedir al model
  LdWidgetModelAbs? get model => _model;

  /// Setter per assignar el model
  @protected
  set model(LdWidgetModelAbs? newModel) {
    if (_model != newModel) {
      // Desregistrar-se de l'antic model si existeix
      if (_model != null) {
        _model!.detachObserver(this);
```

```

    }
    // Assignar el nou model
    _model = newModel;

    // Registrar-se com a observador del nou model
    if (_model != null) {
        _model!.attachObserver(this);
    }

    Debug.info("$tag: Model actualitzat");
}

}

/// Retorna el widget associat al controlador amb el tipus correcte.
T get cWidget => widget;

/// Subscripció als events de l'aplicació
StreamSubscription<LdEvent>? _subcEvent;

/// PROPIETATS DE VISUALITZACIÓ I INTERACCIÓ
=====
/// Flag de visibilitat del widget
bool _isVisible = true;

/// Retorna si el widget és visible
bool get isVisible => _isVisible;

/// Estableix si el widget és visible i reconstrueix la UI si és necessari
set isVisible(bool value) {
    if (_isVisible != value && mounted) {
        setState(() {
            _isVisible = value;
            Debug.info("$tag: Visibilitat canviada a $value");
        });
    } else if (_isVisible != value) {
        // Si no està muntat, només canviem el valor
        _isVisible = value;
        Debug.info("$tag: Visibilitat canviada a $value (sense reconstrucció)");
    }
}

/// Node de focus pel widget
final FocusNode _focusNode = FocusNode();

/// Retorna el node de focus del widget
FocusNode get focusNode => _focusNode;

/// Indica si el widget té el focus
bool get hasFocus => _focusNode.hasFocus;

```

```

    /// Flag que indica si el widget pot rebre focus
    bool _canFocus = true;

    /// Retorna si el widget pot rebre focus
    bool get canFocus => _canFocus;

    /// Estableix si el widget pot rebre focus i reconstrueix la UI si és
necessari
    set canFocus(bool value) {
        if (_canFocus != value && mounted) {
            setState(() {
                _canFocus = value;
                if (!value && hasFocus) {
                    _focusNode.unfocus();
                }
            });
            Debug.info("$tag: Capacitat de focus canviada a $value");
        } else if (_canFocus != value) {
            // Si no està muntat, només canviem el valor
            _canFocus = value;
            if (!value && hasFocus) {
                _focusNode.unfocus();
            }
            Debug.info("$tag: Capacitat de focus canviada a $value (sense
reconstrucció)");
        }
    }

    /// Flag que indica si el widget està actiu (enabled)
    bool _isEnabled = true;

    /// Retorna si el widget està actiu
    bool get isEnabled => _isEnabled;

    /// Estableix si el widget està actiu i reconstrueix la UI si és necessari
    set isEnabled(bool value) {
        if (_isEnabled != value && mounted) {
            setState(() {
                _isEnabled = value;
                if (!value && hasFocus) {
                    _focusNode.unfocus();
                }
            });
            Debug.info("$tag: Estat d'activació canviat a $value");
        } else if (_isEnabled != value) {
            // Si no està muntat, només canviem el valor
            _isEnabled = value;
            if (!value && hasFocus) {
                _focusNode.unfocus();
            }
        }
    }

```

```

        Debug.info("$tag: Estat d'activació canviat a $value (sense
reconstrucció)");
    }
}

/// Demana focus per aquest widget si pot rebre'l i està actiu
void requestFocus() {
    if (canFocus && isEnabled && isVisible) {
        _focusNode.requestFocus();
        Debug.info("$tag: Focus demanat");
    }
}

/// Configura l'estat del widget (visible, focus, enabled) i reconstrueix la
UI
void setWidgetState({
    bool? visible,
    bool? canFocus,
    bool? enabled,
}) {
    bool needsUpdate = false;

    if (visible != null && _isVisible != visible) {
        _isVisible = visible;
        needsUpdate = true;
    }

    if (canFocus != null && _canFocus != canFocus) {
        _canFocus = canFocus;
        if (!canFocus && hasFocus) {
            _focusNode.unfocus();
        }
        needsUpdate = true;
    }

    if (enabled != null && _isEnabled != enabled) {
        _isEnabled = enabled;
        if (!enabled && hasFocus) {
            _focusNode.unfocus();
        }
        needsUpdate = true;
    }

    if (needsUpdate && mounted) {
        setState(() {
            Debug.info("$tag: Estat del widget actualitzat");
        });
    } else if (needsUpdate) {
        Debug.info("$tag: Estat del widget actualitzat (sense reconstrucció)");
    }
}

```

```

    }
}

/// Toggle per canviar la visibilitat
void toggleVisibility() {
    Debug.info("$tag: Cridant a toggleVisibility()");
    if (mounted) {
        setState(() {
            _isVisible = !_isVisible;
            Debug.info("$tag: Visibilitat alternada a $_isVisible");
        });
    } else {
        _isVisible = !_isVisible;
        Debug.info("$tag: Visibilitat alternada a $_isVisible (sense
reconstrucció)");
    }
}

/// Toggle per canviar l'estat actiu
void toggleEnabled() {
    Debug.info("$tag: Cridant a toggleEnabled()");
    if (mounted) {
        setState(() {
            _isEnabled = !_isEnabled;
            Debug.info("$tag: Estat d'activació alternat a $_isEnabled");
        });
    } else {
        _isEnabled = !_isEnabled;
        Debug.info("$tag: Estat d'activació alternat a $_isEnabled (sense
reconstrucció)");
    }
}

/// CONSTRUCTOR -----
LdWidgetCtrlAbs(T pwidget, {
    bool isVisible = true,
    bool canFocus = true,
    bool isEnabled = true
}) {
    // Assignem els valors directament sense cridar setState
    _isVisible = isVisible;
    _canFocus = canFocus;
    _isEnabled = isEnabled;

    // Configurar listener de focus per debugging
    _focusNode.addListener(() {
        if (_focusNode.hasFocus) {
            Debug.info("$tag: Ha rebut el focus");
        } else {
            Debug.info("$tag: Ha perdut el focus");
        }
    });
}

```

```

});
}

/// Inicialitza el controlador
@Override
void initState() {
    super.initState();
    tag = '${widget.tag}_Ctrl';
    _subcEvent = EventBus.s.listen(_handleEvent);

    // Carregar configuració del widget
    _loadConfigFromWidget();

    // Crear el model si el widget ho requereix
    _createModelIfNeeded();

    initialize();
    Debug.info("$tag: Controlador inicialitzat");
}

/// Carrega la configuració des del widget
void _loadConfigFromWidget() {
    final config = widget.config;

    // Carregar propietats del controlador
    _isVisible = config[cfIsVisible] as bool ?? true;
    _canFocus = config[cfCanFocus] as bool ?? true;
    _isEnabled = config[cfIsEnabled] as bool ?? true;

    Debug.info("$tag: Configuració carregada: visible=$_isVisible,
canFocus=$_canFocus, enabled=$_isEnabled");
}

/// Crea el model del widget si és necessari
/// Aquest mètode ha de ser sobreescrit per les subclasses per crear un model
específic
@protected
void _createModelIfNeeded() {
    // Implementació base buida
    // Les subclasses haurien de sobre escriure aquest mètode per crear el seu
model específic
}

/// Processa un event rebut
void _handleEvent(LdEvent event) {
    // Processar events globals o dirigits a aquest controlador
    if (event.eType == EventType.languageChanged ||
        event.eType == EventType.themeChanged ||
        event.eType == EventType.rebuildUI ||
        event.isTargetedAt(tag)) {
        Debug.info("$tag: Processant event ${event.eType.name}");
    }
}

```



```

        onEvent(event);
    }
}

/// Mètode a sobre escriure per gestionar events
void onEvent(LdEvent event);

/// Actualitza el controlador quan canvien les dependències
@Override
void didChangeDependencies() {
    super.didChangeDependencies();
    Debug.info("$tag: Dependències actualitzades");
    update();
}

/// Gestiona l'actualització del widget quan canvien les propietats
@Override
void didUpdateWidget(covariant T oldWidget) {
    super.didUpdateWidget(oldWidget);

    // Verificar si la configuració ha canviat
    if (_hasConfigChanged(oldWidget)) {
        Debug.info("$tag: Configuració actualitzada, recarregant");
        _loadConfigFromWidget();

        // Forçar reconstrucció si és necessari
        setState(() {
            // Reconstruir la UI
        });
    }
}

/// Comprova si la configuració ha canviat
bool _hasConfigChanged(T oldWidget) {
    // Comparar només les propietats del controlador (cf*)
    final oldConfig = oldWidget.config;
    final newConfig = widget.config;

    // Filtrar i comparar només les propietats de controlador (cf*)
    bool hasChanged = false;
    for (final key in newConfig.keys) {
        if (key.startsWith('cf') && key != cfTag) {
            if (oldConfig[key] != newConfig[key]) {
                Debug.info("$tag: Canvi detectat en propietat $key: ${oldConfig[key]}
-> ${newConfig[key]}");
                hasChanged = true;
                break;
            }
        }
    }
}

```

```

    return hasChanged;
}

/// Allibera els recursos
@override
void dispose() {
    Debug.info("$tag: Alliberant recursos del controlador...");

    // Cridar cleanup del widget per alliberar el mapa
    widget.cleanup();

    // Cancelar subscripció a events
    EventBus.s.cancel(_subcEvent);

    // Alliberar recursos del node de focus
    _focusNode.dispose();

    // Desregistrar-se del model
    if (_model != null) {
        _model!.detachObserver(this);
        _model = null;
    }

    super.dispose();
    Debug.info("$tag: Recursos del controlador alliberats");
}

/// Notificació de canvi en un model
@override
void onModelChanged(LdModelAbs pModel, void Function() pfUpdate) {
    Debug.info("$tag: Model ha canviat");

    // Executar la funció d'actualització
    pfUpdate();

    // Reconstruir si està muntat
    if (mounted) {
        setState(() {
            Debug.info("$tag: Reconstruint després del canvi del model");
        });
    }
}

/// Construeix el widget
@override
Widget build(BuildContext context) {
    Debug.info("$tag: Construint widget. isVisible=$_isVisible");

    // Si el widget no és visible, retornem un SizedBox buit

```

```

    if (!_isVisible) {
        return const SizedBox.shrink();
    }

    // Construïm el contingut del widget
    Widget content = buildContent(context);

    // Apliquem l'opacitat i l'absorció d'events si el widget no està actiu
    if (!_isEnabled) {
        content = Opacity(
            opacity: 0.5,
            child: AbsorbPointer(
                absorbing: true,
                child: content,
            ),
        );
    }

    return content;
}

/// Mètode que ha d'implementar cada widget per construir la seva UI
Widget buildContent(BuildContext context);

/// Passarel·la de pas per a la crida a setState().
@override setState(void Function() pFN) {
    super.setState(pFN);
}
}

```

Aquesta implementació completa de `LdWidgetCtrlAbs` inclou:

1. **Gestió de model:** Getter/setter públic amb gestió d'observadors
2. **Propietats de visualització:** Visibilitat, focus i estat actiu
3. **Node de focus:** Gestió completa del focus
4. **Carrega de configuració:** Des del mapa del widget
5. **Gestió d'events:** Subscripció i processament d'events
6. **Cicle de vida Flutter:** Tots els mètodes necessaris
7. **Neteja de recursos:** Alliberament adequat de tots els recursos
8. **Debugging complet:** Logs per a tots els esdeveniments importants

Comprova que aquesta implementació funcioni correctament abans de continuar amb el següent fitxer.

Retry

JI

Aquesta implementació sembla correcte, gràcies. Podem fer el mateix amb 'LdPageCtrlAbs'?

Edit

Aquí tens la implementació completa de `LdPageCtrlAbs` seguint els mateixos principis que `LdWidgetCtrlAbs`:

dart

```
// ld_page_ctrl.dart
// Controlador base per a les pàgines.
// Created: 2025/05/03 ds. JIQ
// Updated: 2025/05/11 ds. CLA - Optimització amb gestió de model i GlobalKey

import 'dart:async';

import 'package:flutter/material.dart';
import 'package:ld_wbench5/core/event_bus/event_bus.dart';
import 'package:ld_wbench5/core/event_bus/ld_event.dart';
import 'package:ld_wbench5/core/ld_model_abs.dart';
import 'package:ld_wbench5/core/ld_page/ld_page_abs.dart';
import 'package:ld_wbench5/core/ld_page/ld_page_model.dart';
import 'package:ld_wbench5/core/ld_taggable_mixin.dart';
import 'package:ld_wbench5/core/lifecycle_interface.dart';
import 'package:ld_wbench5/utils/debug.dart';

/// Controlador base per a les pàgines.
abstract class LdPageCtrl<T> extends LdPageAbs>
    extends State<T>
    with LdTaggableMixin
    implements LdLifecycleIntf, LdModelObserverIntf {

  /// Model de la pàgina
  LdPageModelAbs? _model;

  /// Getter públic per accedir al model
  LdPageModelAbs? get model => _model;

  /// Setter per assignar el model
  @protected
  set model(LdPageModelAbs? newModel) {
    if (_model != newModel) {
      // Desregistrar-se de l'antic model si existeix
      if (_model != null) {
        _model!.detachObserver(this);
      }

      // Assignar el nou model
      _model = newModel;

      // Registrar-se com a observador del nou model
    }
  }
}
```

```

        if (_model != null) {
            _model!.attachObserver(this);
        }
    }

    Debug.info("$tag: Model de pàgina actualitzat");
}

}

/// Referència a la pàgina
T get cPage => widget;

/// Subscripció als events de l'aplicació
StreamSubscription<LdEvent>? _subcEvent;

/// Constructor
LdPageCtrl({ String? pTag, required T pPage }) {
    tag = (pTag != null)
        ? pTag
        : className;
}

/// Inicialitza el controlador
@override
void initState() {
    super.initState();
    tag = '${cPage.tag}_Ctrl';
    _subcEvent = EventBus.s.listen(_handleEvent);

    // Crear el model si la pàgina ho requereix
    _createModelIfNeeded();

    initialize();
    Debug.info("$tag: Controlador de pàgina inicialitzat");
}

/// Crea el model de la pàgina si és necessari
/// Aquest mètode ha de ser sobreescrit per les subclasses per crear un model
específic
@protected
void _createModelIfNeeded() {
    // Implementació base buida
    // Les subclasses haurien de sobre escriure aquest mètode per crear el seu
model específic
}

/// Processa un event rebut
void _handleEvent(LdEvent event) {
    // Eliminar el filtratge per tag en events globals importants
    if (event.eType == EventType.languageChanged ||
        event.eType == EventType.themeChanged ||
        event.eType == EventType.rebuildUI ||

```

```

        event.isTargetedAt(tag)) {
            Debug.info("$tag: Processant event ${event.eType.name}");
            onEvent(event);
        }
    }

    /// Mètode a sobreesciure per gestionar events
    void onEvent(LdEvent event);

    /// Actualitza el controlador quan canvien les dependències
    @override
    void didChangeDependencies() {
        super.didChangeDependencies();
        Debug.info("$tag: Dependències actualitzades");
        update();
    }

    /// Gestiona l'actualització del widget quan canvien les propietats
    @override
    void didUpdateWidget(covariant T oldWidget) {
        super.didUpdateWidget(oldWidget);

        // Verificar si necessitem actualitzar quelcom de la pàgina
        // Per defecte, les pàgines no tenen una configuració tan dinàmica com els
        widgets
        Debug.info("$tag: Pàgina actualitzada");
    }

    /// Allibera els recursos
    @override
    void dispose() {
        Debug.info("$tag: Alliberant recursos de la pàgina...");

        // Cancelar subscripció a events
        EventBus.s.cancel(_subcEvent);

        // Desregistrar-se del model
        if (_model != null) {
            _model!.detachObserver(this);
            _model!.dispose();
            _model = null;
        }

        super.dispose();
        Debug.info("$tag: Recursos de la pàgina alliberats");
    }

    /// Notificació de canvi en un model
    @override
    void onModelChanged(LdModelAbs pModel, void Function() pfUpdate) {

```

```
Debug.info("$tag.onModelChanged(): executant ...");
```

```
// Executar l'actualització sempre
```

```
pfUpdate();
```

```
// Però només reconstruir si està muntat
```

```
if (mounted) {
```

```
    setState(() {
```

```
        Debug.info("$tag.onModelChanged(): Reconstruint widget");
```

```
    });
```

```
}
```

```
Debug.info("$tag.onModelChanged(): ... executat");
```

```
}
```

```
/// Construeix la pàgina
```

```
@override
```

```
Widget build(BuildContext context) {
```

```
    Debug.info("$tag: Construint pàgina");
```

```
    return buildPage(context);
```

```
}
```

```
/// Mètode que ha d'implementar cada pàgina per construir la seva UI
```

```
Widget buildPage(BuildContext context);
```

```
// UTILITATS DE NAVEGACIÓ I CONTEXT =====
```

```
// Aquestes utilitats hereten del mixin LdTaggableMixin
```

```
/// Accés directe al context de la pàgina
```

```
BuildContext get pageContext => context;
```

```
/// Mostra un SnackBar específic de la pàgina
```

```
void showPageSnackBar(String message, {Duration? duration}) {
```

```
    ScaffoldMessenger.of(pageContext).showSnackBar(
```

```
        SnackBar(
```

```
            content: Text(message),
```

```
            duration: duration ?? const Duration(seconds: 3),
```

```
        ),
```

```
    );
```

```
}
```

```
/// Mostra un diàleg modal des de la pàgina
```

```
Future<T?> showPageDialog<T>({
```

```
    required Widget Function(BuildContext) builder,
```

```
    bool barrierDismissible = true,
```

```
}) async {
```

```
    return await showDialog<T>(  
        context: pageContext,
```

```
        barrierDismissible: barrierDismissible,
```

```
        builder: builder,
```

```
    );
```

```

    }
}

/// Tanca la pàgina actual
void closePage([dynamic result]) {
    Navigator.of(pageContext).pop(result);
}

/// Navega a una nova pàgina amb animació
void navigateToPage(
    Widget destination, {
    bool replace = false,
    bool clearStack = false,
    RouteTransitionsBuilder? transitionBuilder,
}) {
    Route<dynamic> route;

    if (transitionBuilder != null) {
        route = PageRouteBuilder(
            pageBuilder: (context, animation, secondaryAnimation) => destination,
            transitionsBuilder: transitionBuilder,
        );
    } else {
        route = MaterialPageRoute(builder: (_) => destination);
    }

    if (clearStack) {
        Navigator.of(pageContext).pushAndRemoveUntil(route, (route) => false);
    } else if (replace) {
        Navigator.of(pageContext).pushReplacement(route);
    } else {
        Navigator.of(pageContext).push(route);
    }
}

/// Mostra un bottom sheet des de la pàgina
Future<T?> showPageBottomSheet<T>({
    required Widget Function(BuildContext) builder,
    bool isScrollControlled = false,
    bool useRootNavigator = false,
}) async {
    return await showModalBottomSheet<T>(
        context: pageContext,
        isScrollControlled: isScrollControlled,
        useRootNavigator: useRootNavigator,
        builder: builder,
    );
}
}

```

Les característiques principals d'aquesta implementació:

1. **Gestió de model:** Similar a `LdWidgetCtrlAbs` però específic per a models de pàgina
2. **Accessors específics:** `cPage` en lloc de `cWidget`
3. **Navegació mejorada:** Mètodes específics per a navegació entre pàgines
4. **Utilitats de UI:** `SnackBars`, diàlegs i bottom sheets
5. **Simplificació:** Sense gestió de focus/visibilitat que és més específica de widgets
6. **Events globals:** Gestió adequada per a elements que sempre estan actius
7. **Neteja de recursos:** Alliberament automàtic del model

La diferència principal amb els widgets és que les pàgines:

- No gestionen propietats com visibilitat, focus o estat actiu
- Tenen mètodes de navegació més sofisticats
- Tenen un cicle de vida més simple i estable
- Poden tenir models més complexos i persistents

Comprova que aquesta implementació funcioni correctament abans de continuar amb el següent fitxer.

Retry

JI

També deu ser necessari revisar 'LdPageAbs', oi?

Edit

Sí, absolutament! Hem de revisar `LdPageAbs` per fer-la coherent amb les optimitzacions que hem fet als altres fitxers. Aquí tens la implementació actualitzada:

dart

```
// ld_page_abs.dart
// Pàgina base simplificada per a l'aplicació
// Created: 2025/04/29 dt. CLA[JIQ]
// Updated: 2025/05/11 ds. CLA - Optimització amb GlobalKey transparent

import 'package:flutter/material.dart';

import 'package:ld_wbench5/core/ld_model_abs.dart';
import 'package:ld_wbench5/core/ld_page/ld_page_ctrl.dart';
import 'package:ld_wbench5/core/ld_page/ld_page_model.dart';
import 'package:ld_wbench5/core/ld_taggable_mixin.dart';
import 'package:ld_wbench5/utills/debug.dart';

export 'ld_page_ctrl.dart';
export 'ld_page_model.dart';

/// Pàgina base que proporciona funcionalitats comunes
```

```

abstract class LdPageAbs extends StatefulWidget
  with LdTaggableMixin
  implements LdModelObserverIntf {
  /// Constructor
  LdPageAbs({
    String? pTag,
  }) : super(key: null) {
    tag = pTag ?? className;
    Debug.info("$tag: Creant pàgina");
  }

  /// Sobreescrivim key per retornar la GlobalKey sota demanda
  @override
  Key? get key {
    if (globalKey == null) {
      // Crear la GlobalKey amb el tipus genèric adequat
      forceGlobalKey<LdPageCtrl>();
    }
    return globalKey;
  }

  // ACCÉS AL CONTROLADOR =====
  /// Retorna el controlador de la pàgina utilitzant la GlobalKey
  LdPageCtrl? get vCtrl {
    if (globalKey == null) {
      // Forçar la creació de la GlobalKey
      forceGlobalKey<LdPageCtrl>();
    }
    return getState<LdPageCtrl>();
  }

  /// Retorna el controlador de la pàgina (versió que garanteix non-null)
  LdPageCtrl get vCtrlRequired {
    final ctrl = vCtrl;
    assert(ctrl != null, "$tag: Controlador de pàgina no disponible encara");
    return ctrl!;
  }

  // ACCÉS AL MODEL (delegat al controlador) =====
  /// Retorna el model de la pàgina (accés a través del controlador)
  LdPageModelAbs? get vModel => vCtrl?.model;

  /// Retorna el model de la pàgina (versió que garanteix non-null)
  LdPageModelAbs get vModelRequired {
    final model = vModel;
    assert(model != null, "$tag: Model de pàgina no disponible encara");
    return model!;
  }
}

```

```
/// Indica si la pàgina té un model assignat
bool get hasModel => vModel != null;
```

```
/// OBSERVADOR DEL MODEL =====
```

```
/// Implementació del LdModelObserverIntf
```

```
@override
```

```
void onModelChanged(LdModelAbs model, void Function() pfUpdate) {
```

```
    // Delegar al controlador quan estigui disponible
```

```
    final ctrl = vCtrl;
```

```
    if (ctrl != null) {
```

```
        ctrl.onModelChanged(model, pfUpdate);
```

```
    } else {
```

```
        // Si el controlador no està disponible encara,
```

```
        // només executar la funció d'actualització
```

```
        pfUpdate();
```

```
    }
```

```
}
```

```
/// CREACIÓ DE CONTROLADOR =====
```

```
/// Mètode que cada pàgina ha d'implementar per crear el seu controlador
```

```
@protected
```

```
LdPageCtrl createController();
```

```
/// Retorna el controlador de la pàgina
```

```
@override
```

```
State<LdPageAbs> createState() => createController();
```

```
}
```

Els canvis principals que he fet a **LdPageAbs**:

1. **Eliminació de OnceSet**: Ja no utilitzem OnceSet per emmagatzemar el controlador i el model
2. **GlobalKey transparent**: La GlobalKey es crea sota demanda com als widgets
3. **Accés unificat**: Els accessors són similars als dels widgets
(wCtrl/wModel vs vCtrl/vModel)
4. **Delegació al controlador**: El model es gestiona completament pel controlador
5. **Mètode createController**: Cada pàgina ha d'implementar aquest mètode per crear el seu controlador
6. **Similat arquitectura**: Ara segueix la mateixa estructura que els widgets

Exemple d'ús actualitzat:

dart

```
// Exemple de pàgina actualitzada
```

```
class TestPage extends LdPageAbs {
```

```
    TestPage({
```

```
        super.pTag,
```

```

        required String pTitleKey,
        String? pSubTitleKey,
    });
}

@override
LdPageCtrl createController() {
    // Crear el controlador
    final ctrl = TestPageCtrl(pPage: this);

    // Crear i assignar el model al controlador
    ctrl.model = TestPageModel(
        pPage: this,
        pTitleKey: pTitleKey,
        pSubTitleKey: pSubTitleKey,
    );

    return ctrl;
}
}

```

Aquesta implementació fa que les pàgines siguin consistents amb els widgets en termes de:

- Gestió de GlobalKey
- Accés al controlador i model
- Cicle de vida i recursos
- Implementació simplificada

Comprova que aquesta implementació funcioni correctament abans de continuar amb la refactorització de widgets específics.

Retry

JI

1. 'Future<T?> showPageDialog<T>' i 'Future<T?> showPageBottomSheet<T>' oculten un tipus 'T' superior en el codi de 'LdPageCtrl'.
2. 'LdPageCtrl' s'ha d'anomenar 'LdPageCtrlAbs' perquè és el criteri per a identificar pel nom d'una classe si aquesta és abstracta o no.

Edit

Tens raó en les dues observacions. Anem a corregir aquests problemes:

dart

```

// ld_page_ctrl_abs.dart
// Controlador base abstracte per a les pàgines.
// Created: 2025/05/03 ds. JIQ
// Updated: 2025/05/11 ds. CLA - Correcció de noms i tipus genèrics

```

```

import 'dart:async';

import 'package:flutter/material.dart';
import 'package:ld_wbench5/core/event_bus/event_bus.dart';
import 'package:ld_wbench5/core/event_bus/ld_event.dart';
import 'package:ld_wbench5/core/ld_model_abs.dart';
import 'package:ld_wbench5/core/ld_page/ld_page_abs.dart';
import 'package:ld_wbench5/core/ld_page/ld_page_model.dart';
import 'package:ld_wbench5/core/ld_taggable_mixin.dart';
import 'package:ld_wbench5/core/lifecycle_interface.dart';
import 'package:ld_wbench5/utils/debug.dart';

/// Controlador base abstracte per a les pàgines.
abstract class LdPageCtrlAbs<T extends LdPageAbs>
    extends State<T>
    with LdTaggableMixin
    implements LdLifecycleIntf, LdModelObserverIntf {

    /// Model de la pàgina
    LdPageModelAbs? _model;

    /// Getter públic per accedir al model
    LdPageModelAbs? get model => _model;

    /// Setter per assignar el model
    @protected
    set model(LdPageModelAbs? newModel) {
        if (_model != newModel) {
            // Desregistrar-se de l'antic model si existeix
            if (_model != null) {
                _model!.detachObserver(this);
            }

            // Assignar el nou model
            _model = newModel;

            // Registrar-se com a observador del nou model
            if (_model != null) {
                _model!.attachObserver(this);
            }

            Debug.info("$tag: Model de pàgina actualitzat");
        }
    }

    /// Referència a la pàgina
    T get cPage => widget;

    /// Subscripció als events de l'aplicació
    StreamSubscription<LdEvent>? _subcEvent;

```

```

    /// Constructor
    LdPageCtrlAbs({ String? pTag, required T pPage }) {
        tag = (pTag != null)
            ? pTag
            : className;
    }

    /// Inicialitza el controlador
    @override
    void initState() {
        super.initState();
        tag = '${cPage.tag}_Ctrl';
        _subcEvent = EventBus.s.listen(_handleEvent);

        // Crear el model si la pàgina ho requereix
        _createModelIfNeeded();

        initialize();
        Debug.info("$tag: Controlador de pàgina inicialitzat");
    }

    /// Crea el model de la pàgina si és necessari
    /// Aquest mètode ha de ser sobreescrit per les subclasses per crear un model
    específic
    @protected
    void _createModelIfNeeded() {
        // Implementació base buida
        // Les subclasses haurien de sobreescriure aquest mètode per crear el seu
        model específic
    }

    /// Processa un event rebut
    void _handleEvent(LdEvent event) {
        // Eliminar el filtratge per tag en events globals importants
        if (event.eType == EventType.languageChanged ||
            event.eType == EventType.themeChanged ||
            event.eType == EventType.rebuildUI ||
            event.isTargetedAt(tag)) {
            Debug.info("$tag: Processant event ${event.eType.name}");
            onEvent(event);
        }
    }

    /// Mètode a sobreescriure per gestionar events
    void onEvent(LdEvent event);

    /// Actualitza el controlador quan canvien les dependències
    @override
    void didChangeDependencies() {
        super.didChangeDependencies();
    }

```

```

    Debug.info("$tag: Dependències actualitzades");
    update();
}

/// Gestiona l'actualització del widget quan canvien les propietats
@Override
void didUpdateWidget(covariant T oldWidget) {
    super.didUpdateWidget(oldWidget);

    // Verificar si necessitem actualitzar quelcom de la pàgina
    // Per defecte, les pàgines no tenen una configuració tan dinàmica com els
    widgets
    Debug.info("$tag: Pàgina actualitzada");
}

/// Allibera els recursos
@Override
void dispose() {
    Debug.info("$tag: Alliberant recursos de la pàgina...");

    // Cancel·lar subscripció a events
    EventBus.s.cancel(_subcEvent);

    // Desregistrar-se del model
    if (_model != null) {
        _model!.detachObserver(this);
        _model!.dispose();
        _model = null;
    }

    super.dispose();
    Debug.info("$tag: Recursos de la pàgina alliberats");
}

/// Notificació de canvi en un model
@Override
void onModelChanged(LdModelAbs pModel, void Function() pfUpdate) {
    Debug.info("$tag.onModelChanged(): executant ...");

    // Executar l'actualització sempre
    pfUpdate();

    // Però només reconstruir si està muntat
    if (mounted) {
        setState(() {
            Debug.info("$tag.onModelChanged(): Reconstruint widget");
        });
    }

    Debug.info("$tag.onModelChanged(): ... executat");
}

```

```

}

/// Construeix la pàgina
@override
Widget build(BuildContext context) {
  Debug.info("$tag: Construïnt pàgina");
  return buildPage(context);
}

/// Mètode que ha d'implementar cada pàgina per construir la seva UI
Widget buildPage(BuildContext context);

// UTILITATS DE NAVEGACIÓ I CONTEXT =====
// Aquestes utilitats hereten del mixin LdTaggableMixin

/// Accés directe al context de la pàgina
BuildContext get pageContext => context;

/// Mostra un SnackBar específic de la pàgina
void showPageSnackBar(String message, {Duration? duration}) {
  ScaffoldMessenger.of(pageContext).showSnackBar(
    SnackBar(
      content: Text(message),
      duration: duration ?? const Duration(seconds: 3),
    ),
  );
}

/// Mostra un diàleg modal des de la pàgina
Future<R?> showPageDialog<R>({
  required Widget Function(BuildContext) builder,
  bool barrierDismissible = true,
}) async {
  return await showDialog<R>(
    context: pageContext,
    barrierDismissible: barrierDismissible,
    builder: builder,
  );
}

/// Tanca la pàgina actual
void closePage([dynamic result]) {
  Navigator.of(pageContext).pop(result);
}

/// Navega a una nova pàgina amb animació
void navigateToPage(
  Widget destination, {
  bool replace = false,
  bool clearStack = false,
  RouteTransitionsBuilder? transitionBuilder,

```



```

    }) {
      Route<dynamic> route;

      if (transitionBuilder != null) {
        route = PageRouteBuilder(
          pageBuilder: (context, animation, secondaryAnimation) => destination,
          transitionsBuilder: transitionBuilder,
        );
      } else {
        route = MaterialPageRoute(builder: (_) => destination);
      }

      if (clearStack) {
        Navigator.of(pageContext).pushAndRemoveUntil(route, (route) => false);
      } else if (replace) {
        Navigator.of(pageContext).pushReplacement(route);
      } else {
        Navigator.of(pageContext).push(route);
      }
    }
  }

  /// Mostra un bottom sheet des de la pàgina
  Future<R?> showPageBottomSheet<R>({
    required Widget Function(BuildContext) builder,
    bool isScrollControlled = false,
    bool useRootNavigator = false,
  }) async {
    return await showModalBottomSheet<R>({
      context: pageContext,
      isScrollControlled: isScrollControlled,
      useRootNavigator: useRootNavigator,
      builder: builder,
    });
  }
}

```

Els canvis realitzats:

1. **Canvi de nom:** `LdPageCtrl` → `LdPageCtrlAbs` per mantenir la coherència de nomenclatura
2. **Tipus genèrics corregits:**

- `showPageDialog<R>` i `showPageBottomSheet<R>` utilitzen `R` en lloc de `T`
- Això evita el conflicte amb el tipus genèric `T` de la classe

També hem de actualitzar `LdPageAbs` per referenciar correctament el nom canviat:

dart

```

// ld_page_abs.dart
// Actualització per referenciar LdPageCtrlAbs
// Updated: 2025/05/11 ds. CLA - Correcció de referència a controlador

import 'package:flutter/material.dart';

import 'package:ld_wbench5/core/ld_model_abs.dart';
import 'package:ld_wbench5/core/ld_page/ld_page_ctrl_abs.dart';
import 'package:ld_wbench5/core/ld_page/ld_page_model.dart';
import 'package:ld_wbench5/core/ld_taggable_mixin.dart';
import 'package:ld_wbench5/utils/debug.dart';

export 'ld_page_ctrl_abs.dart';
export 'ld_page_model.dart';

/// Pàgina base que proporciona funcionalitats comunes
abstract class LdPageAbs extends StatefulWidget
    with LdTaggableMixin
    implements LdModelObserverIntf {

  /// Constructor
  LdPageAbs({
    String? pTag,
  }) : super(key: null) {
    tag = pTag ?? className;
    Debug.info("$tag: Creant pàgina");
  }

  /// Sobreescrivim key per retornar la GlobalKey sota demanda
  @override
  Key? get key {
    if (globalKey == null) {
      // Crear la GlobalKey amb el tipus genèric adequat
      forceGlobalKey<LdPageCtrlAbs>();
    }
    return globalKey;
  }

  // ACCÉS AL CONTROLADOR =====
  /// Retorna el controlador de la pàgina utilitzant la GlobalKey
  LdPageCtrlAbs? get vCtrl {
    if (globalKey == null) {
      // Forçar la creació de la GlobalKey
      forceGlobalKey<LdPageCtrlAbs>();
    }
    return getState<LdPageCtrlAbs>();
  }

  /// Retorna el controlador de la pàgina (versió que garanteix non-null)
  LdPageCtrlAbs get vCtrlRequired {
    final ctrl = vCtrl;

```

```

    assert(ctrl != null, "$tag: Controlador de pàgina no disponible encara");
    return ctrl!;
}

// ACCÉS AL MODEL (delegat al controlador) =====
/// Retorna el model de la pàgina (accés a través del controlador)
LdPageModelAbs? get vModel => vCtrl?.model;

/// Retorna el model de la pàgina (versió que garanteix non-null)
LdPageModelAbs get vModelRequired {
    final model = vModel;
    assert(model != null, "$tag: Model de pàgina no disponible encara");
    return model!;
}

/// Indica si la pàgina té un model assignat
bool get hasModel => vModel != null;

// OBSERVADOR DEL MODEL =====
/// Implementació del LdModelObserverIntf
@override
void onModelChanged(LdModelAbs model, void Function() pfUpdate) {
    // Delegar al controlador quan estigui disponible
    final ctrl = vCtrl;
    if (ctrl != null) {
        ctrl.onModelChanged(model, pfUpdate);
    } else {
        // Si el controlador no està disponible encara,
        // només executar la funció d'actualització
        pfUpdate();
    }
}

/// CREACIÓ DE CONTROLADOR =====
/// Mètode que cada pàgina ha d'implementar per crear el seu controlador
@protected
LdPageCtrlAbs createController();

/// Retorna el controlador de la pàgina
@override
State<LdPageAbs> createState() => createController();
}

```

Ara aquestes implementacions són consistents i coherents amb el patró de nomenclatura del projecte.

Retry

Jl

Ja he actualitzat i revisat 'LdPageAbs'. Tot sembla anar com tenim previst. Continuem amb el següent fitxer.

Edit

Perfecte! El següent fitxer a actualitzar és `LdWidgetModelAbs`. Aquesta classe base ha de ser adaptada per funcionar amb el nostre nou sistema de mapes:

dart

```
// ld_widget_model_abs.dart
// Model base pels widgets.
// Created: 2025/05/02 dj. JIQ
// Updated: 2025/05/11 ds. CLA - Optimització per mapes i arquitectura
simplificada

import 'package:ld_wbench5/core/ld_model_abs.dart';
import 'package:ld_wbench5/core/ld_widget/ld_widget_abs.dart';
import 'package:ld_wbench5/core/map_fields.dart';
import 'package:ld_wbench5/ui/extensions/map_extensions.dart';
import 'package:ld_wbench5/utils/debug.dart';

/// Model base pels widgets.
abstract class LdWidgetModelAbs<T extends LdWidgetAbs> extends LdModelAbs {
  /// Construeix un model a partir d'un mapa de propietats
  LdWidgetModelAbs.fromMap(LdMap<dynamic> pMap) {
    // Filtrar només les propietats que comencen amb 'mf'
    LdMap<dynamic> modelProperties = {};
    for (var entry in pMap.entries) {
      if (entry.key.startsWith('mf')) {
        modelProperties[entry.key] = entry.value;
      }
    }
  }

  // Establir el tag des del mapa (prioritzant mfTag, després cfTag)
  tag = pMap[mfTag] as String? ??
    pMap[cfTag] as String? ??
    "Model_${DateTime.now().millisecondsSinceEpoch}";

  // Carregar les propietats filtrades
  fromMap(modelProperties);

  Debug.info("$tag: Model creat a partir de mapa");
}

/// Constructor alternatiu amb widget específic
/// Utilitzat quan el widget específic necessita configuració especial
@protected
LdWidgetModelAbs.forWidget(T pWidget, LdMap<dynamic> pMap) {
```

```

    // Filtrar propietats del model (mf*)
    LdMap<dynamic> modelProperties = {};
    for (var entry in pMap.entries) {
        if (entry.key.startsWith('mf')) {
            modelProperties[entry.key] = entry.value;
        }
    }

    // Establir el tag basat en el widget
    tag = "${pWidget.tag}_Model";

    // Carregar les propietats
    if (modelProperties.isNotEmpty) {
        fromMap(modelProperties);
    }

    Debug.info("$tag: Model creat per al widget ${pWidget.tag}");
}

/// Constructor obsolet
@deprecated
LdWidgetModelAbs(T pWidget) {
    Debug.warn("$tag: Utilitzant constructor obsolet. Canvieu al constructor
amb mapa de propietats.");
    tag = "${pWidget.tag}_Model";
}

/// Converteix el model a un mapa de propietats
@override
LdMap<dynamic> toMap() {
    LdMap<dynamic> map = super.toMap();
    map[mfTag] = tag;
    return map;
}

/// Carrega el model des d'un mapa de propietats
@override
void fromMap(LdMap<dynamic> pMap) {
    super.fromMap(pMap);
    // Específic de cada implementació
}

/// Actualitza només les propietats específiques del model
void updateFromMap(LdMap<dynamic> pMap) {
    // Filtrar només les propietats que comencen amb 'mf'
    LdMap<dynamic> modelProperties = {};
    for (var entry in pMap.entries) {
        if (entry.key.startsWith('mf')) {
            modelProperties[entry.key] = entry.value;
        }
    }
}

```

```

    }
}

// Actualitzar només si hi ha propietats
if (modelProperties.isNotEmpty) {
    notifyListeners(() {
        fromMap(modelProperties);
    });
    Debug.info("$tag: Model actualitzat amb noves propietats");
}
}

/// Actualitza una propietat específica del model
void updateField(String fieldKey, dynamic value) {
    // Verificar que la clau comença amb 'mf'
    if (!fieldKey.startsWith('mf')) {
        Debug.warn("$tag: Intent d'actualitzar camp que no és de model: $fieldKey");
        return;
    }
    notifyListeners(() {
        setField(pKey: fieldKey, pValue: value);
    });
    Debug.info("$tag: Camp actualitzat: $fieldKey = $value");
}

/// Obté múltiples camps del model
LdMap<dynamic> getFields(List<String> fieldKeys) {
    LdMap<dynamic> result = {};
    for (String key in fieldKeys) {
        if (key.startsWith('mf')) {
            try {
                result[key] = getField(pKey: key);
            } catch (e) {
                Debug.warn("$tag: No s'ha pogut obtenir el camp $key: $e");
            }
        }
    }
    return result;
}

/// Actualitza múltiples camps del model alhora
void setFields(LdMap<dynamic> fields) {
    // Filtrar només camps del model
    LdMap<dynamic> modelFields = {};
    for (var entry in fields.entries) {
        if (entry.key.startsWith('mf')) {

```

```

        modelFields[entry.key] = entry.value;
    }
}

if (modelFields.isNotEmpty) {
    notifyListeners() {
        for (var entry in modelFields.entries) {
            setField(pKey: entry.key, pValue: entry.value);
        }
    });
}

Debug.info("$tag: Múltiples camps actualitzats: ${modelFields.keys.toList()}");
}
}

/// Copia el model a un nou mapa amb només les propietats de model
LdMap<dynamic> toModelMap() {
    final fullMap = toMap();
    LdMap<dynamic> modelOnly = {};

    for (var entry in fullMap.entries) {
        if (entry.key.startsWith('mf')) {
            modelOnly[entry.key] = entry.value;
        }
    }

    return modelOnly;
}

/// Compara aquest model amb un altre mapa per detectar canvis
bool hasChangesFrom(LdMap<dynamic> otherMap) {
    final currentMap = toModelMap();

    // Comparar claus existents
    for (var key in currentMap.keys) {
        if (!otherMap.containsKey(key) || currentMap[key] != otherMap[key]) {
            return true;
        }
    }

    // Comparar claus noves en otherMap
    for (var key in otherMap.keys) {
        if (key.startsWith('mf') && !currentMap.containsKey(key)) {
            return true;
        }
    }

    return false;
}
}

```

```
}
```

Els canvis principals en aquesta implementació:

1. **Constructor principal `fromMap`**: Filtra automàticament propietats del model (mf*)
2. **Constructor alternatiu `forWidget`**: Per a casos on necessitem referència al widget
3. **Mètodes d'actualització:**

- `updateFromMap`: Actualitza el model des d'un mapa
- `updateField`: Actualitza un camp específic
- `setFields`: Actualitza múltiples camps alhora

4. **Mètodes de lectura:**

- `getFields`: Obté múltiples camps alhora
- `toModelMap`: Converteix a mapa només amb propietats del model

5. **Utilitats:**

- `hasChangesFrom`: Compara amb un altre mapa per detectar canvis

6. **Gestió automàtica de notifikacions:** Tots els canvis notifiquen als observadors

Aquesta implementació fa que els models siguin més flexibles i fàcils d'utilitzar amb el sistema de mapes que estem implementant.

Comprova que aquesta implementació funcioni correctament abans de continuar amb el següent fitxer.

Retry

JI

Per què em dona un error de compilació posar-li un missatge de deprecació a aquest codi?:

```
@deprecated("Fes servir els constructors 'fromMap' o 'forWidget' enlloc d'aquest antic.")
```

Edit

L'error de compilació és perquè `@deprecated` només accepta un String directament, no una cadena amb cometes dins. La sintaxi correcta seria:

```
dart
```

```
@deprecated
```

```
LdWidgetModelAbs(T pwidget) {
```

```
    Debug.warn("$tag: Utilitzant constructor obsolet. Canvieu al constructor amb  
mapa de propietats.");
```

```
    tag = "${pwidget.tag}_Model";
```

```
}
```


O si vols posar un missatge personalitzat:

dart

```
@Deprecated("Fes servir els constructors 'fromMap' o 'forWidget' enlloc  
d'aquest antic.")  
LdWidgetModelAbs(T pwidget) {  
  Debug.warn("$tag: Utilitzant constructor obsolet. Canvieu al constructor amb  
mapa de propietats.");  
  tag = "${pwidget.tag}_Model";  
}
```

La diferència és que `@deprecated` és una anotació que no accepta paràmetres, mentre que `@Deprecated()` (amb "D" majúscula) és una classe que sí que pot acceptar un missatge personalitzat entre cometes.

Les opcions correctes són:

1. `@deprecated` (sense missatge)
2. `@Deprecated("El teu missatge aquí")`

Però no pots fer `@deprecated("missatge")` perquè la anotació `deprecated` en minúscules no accepta paràmetres.

Retry

JI

Gràcies, entesos.

Continuem amb el proper fitxer.

Edit

Perfecte! El següent fitxer a actualitzar és `LdPageModelAbs`. Aquest model base per a pàgines ha de seguir la mateixa estructura que `LdWidgetModelAbs` però adaptada per a pàgines:

dart

```
// ld_page_model.dart  
// Model base per a les pàgines.  
// Created: 2025/05/03 ds. JIQ  
// Updated: 2025/05/11 ds. CLA - Optimització per mapes i arquitectura  
simplificada
```

```
import 'package:ld_wbench5/core/ld_model_abs.dart';  
import 'package:ld_wbench5/core/ld_page/ld_page_abs.dart';  
import 'package:ld_wbench5/core/map_fields.dart';  
import 'package:ld_wbench5/ui/extensions/map_extensions.dart';  
import 'package:ld_wbench5/utills/debug.dart';
```

```

/// Model base per a les pàgines.
abstract class LdPageModelAbs<T extends LdPageAbs> extends LdModelAbs {
    /// Construeix un model a partir d'un mapa de propietats
    LdPageModelAbs.fromMap(LdMap<dynamic> pMap) {
        // Filtrar només les propietats que comencen amb 'mf'
        LdMap<dynamic> modelProperties = {};
        for (var entry in pMap.entries) {
            if (entry.key.startsWith('mf')) {
                modelProperties[entry.key] = entry.value;
            }
        }

        // Establir el tag des del mapa (prioritzant mfTag, després cfTag)
        tag = pMap[mfTag] as String? ??
            pMap[cfTag] as String? ??
            "PageModel_${DateTime.now().millisecondsSinceEpoch}";

        // Carregar les propietats filtrades
        fromMap(modelProperties);

        Debug.info("$tag: Model de pàgina creat a partir de mapa");
    }

    /// Constructor alternatiu amb pàgina específica
    /// Utilitzat quan la pàgina específica necessita configuració especial
    @protected
    LdPageModelAbs.forPage(T pPage, LdMap<dynamic> pMap) {
        // Filtrar propietats del model (mf*)
        LdMap<dynamic> modelProperties = {};
        for (var entry in pMap.entries) {
            if (entry.key.startsWith('mf')) {
                modelProperties[entry.key] = entry.value;
            }
        }

        // Establir el tag basat en la pàgina
        tag = "${pPage.tag}_Model";

        // Carregar les propietats
        if (modelProperties.isNotEmpty) {
            fromMap(modelProperties);
        }

        Debug.info("$tag: Model creat per a la pàgina ${pPage.tag}");
    }

    /// Constructor obsolet
    @Deprecated("Fes servir els constructors 'fromMap' o 'forPage' enlloc d'aquest antic.")

```

```

LdPageModelAbs({ required T pPage }) {
    Debug.warn("$tag: Utilitzant constructor obsolet. Canvieu al constructor
amb mapa de propietats.");
    tag = "${pPage.tag}_Model";
}

/// Converteix el model a un mapa de propietats
@Override
LdMap<dynamic> toMap() {
    LdMap<dynamic> map = super.toMap();
    map[mfTag] = tag;
    return map;
}

/// Carrega el model des d'un mapa de propietats
@Override
void fromMap(LdMap<dynamic> pMap) {
    super.fromMap(pMap);
    // Específic de cada implementació
}

/// Actualitza només les propietats específiques del model
void updateFromMap(LdMap<dynamic> pMap) {
    // Filtrar només les propietats que comencen amb 'mf'
    LdMap<dynamic> modelProperties = {};
    for (var entry in pMap.entries) {
        if (entry.key.startsWith('mf')) {
            modelProperties[entry.key] = entry.value;
        }
    }

    // Actualitzar només si hi ha propietats
    if (modelProperties.isNotEmpty) {
        notifyListeners(() {
            fromMap(modelProperties);
        });
        Debug.info("$tag: Model de pàgina actualitzat amb noves propietats");
    }
}

/// Actualitza una propietat específica del model
void updateField(String fieldKey, dynamic value) {
    // Verificar que la clau comença amb 'mf'
    if (!fieldKey.startsWith('mf')) {
        Debug.warn("$tag: Intent d'actualitzar camp que no és de model:
$fieldKey");
        return;
    }

    notifyListeners(() {

```

```

        setField(pKey: fieldKey, pValue: value);
    });
}

Debug.info("$tag: Camp actualitzat: $fieldKey = $value");
}

/// Obté múltiples camps del model
LdMap<dynamic> getFields(List<String> fieldKeys) {
    LdMap<dynamic> result = {};

    for (String key in fieldKeys) {
        if (key.startsWith('mf')) {
            try {
                result[key] = getField(pKey: key);
            } catch (e) {
                Debug.warn("$tag: No s'ha pogut obtenir el camp $key: $e");
            }
        }
    }

    return result;
}

/// Actualitza múltiples camps del model alhora
void setFields(LdMap<dynamic> fields) {
    // Filtrar només camps del model
    LdMap<dynamic> modelFields = {};

    for (var entry in fields.entries) {
        if (entry.key.startsWith('mf')) {
            modelFields[entry.key] = entry.value;
        }
    }

    if (modelFields.isNotEmpty) {
        notifyListeners(() {
            for (var entry in modelFields.entries) {
                setField(pKey: entry.key, pValue: entry.value);
            }
        });
    }

    Debug.info("$tag: Múltiples camps actualitzats: ${modelFields.keys.toList()}");
}

/// Copia el model a un nou mapa amb només les propietats de model
LdMap<dynamic> toModelMap() {
    final fullMap = toMap();
    LdMap<dynamic> modelOnly = {};
}

```

```

    for (var entry in fullMap.entries) {
        if (entry.key.startsWith('mf')) {
            modelOnly[entry.key] = entry.value;
        }
    }

    return modelOnly;
}

/// Compara aquest model amb un altre mapa per detectar canvis
bool hasChangesFrom(LdMap<dynamic> otherMap) {
    final currentMap = toModelMap();

    // Comparar claus existents
    for (var key in currentMap.keys) {
        if (!otherMap.containsKey(key) || currentMap[key] != otherMap[key]) {
            return true;
        }
    }

    // Comparar claus noves en otherMap
    for (var key in otherMap.keys) {
        if (key.startsWith('mf') && !currentMap.containsKey(key)) {
            return true;
        }
    }

    return false;
}

/// Valida el model abans de persistir-lo
bool validate() {
    // Implementació base - pot ser sobreescrit per validacions específiques
    return true;
}

/// Mètode per persistir el model (pot connectar amb backend)
Future<bool> save() async {
    if (!validate()) {
        Debug.warn("$tag: Validació fallida, no es pot guardar el model");
        return false;
    }

    // Implementació base - pot ser sobreescrit per lògica de persistència
    // específica
    Debug.info("$tag: Model de pàgina guardat");
    return true;
}

/// Mètode per restaurar el model des de persistència

```

```
Future<bool> load(String identifier) async {
  // Implementació base - pot ser sobreescrit per lògica de càrrega
  específica
  Debug.info("$tag: Model de pàgina carregat amb identificador:
  $identifier");
  return true;
}
}
```

Les diferències principals entre `LdPageModelAbs` i `LdWidgetModelAbs`:

1. **Nomenclatura:** Tots els mètodes i missatges fan referència a "pàgina" en lloc de "widget"
2. **Constructor** `forPage`: Equivalent a `forWidget` però per a pàgines
3. **Mètodes addicionals:**

- `validate()`: Per validar dades abans de persistir
- `save()`: Per guardar el model (pot connectar amb backend)
- `load()`: Per carregar el model des de persistència

4. **Mateix paradigma:** Mateixa estructura i funcionalitats base que els models de widget

Aquesta implementació fa que els models de pàgina siguin consistents amb els models de widget mentre proporcionen funcionalitats addicionals que són més útils per a pàgines completes que solen tenir cicles de vida més llargs i poden necessitar persistència.

Comprova que aquesta implementació funcioni correctament abans de continuar amb el següent fitxer.

Retry

JI

Excel·lents afegits! molt bé!

L'he actualitzat i revisat. Em sembla que tot està correcte. Podem continuar.

Edit

Fantàstic! Ara actualitzarem `ModelService` per canviar-li el nom a `MapsService` com vam acordar:

dart

```
// lib/services/maps_service.dart
// Servei centralitzat de gestió de mapes de dades.
// Created: 2025/05/09 dv. CLA[JIQ]
// Updated: 2025/05/11 ds. CLA - Canvi de nom de ModelService a MapsService

import 'package:ld_wbench5/ui/extensions/map_extensions.dart';
import 'package:ld_wbench5/utils/debug.dart';
```

```

/// Servei centralitzat de gestió de mapes de dades.
class MapsService {
    /// Instància singleton
    static final MapsService _inst = MapsService._();
    static MapsService get s => _inst;

    // Membres de la instància -----
    /// Map de mapes indexat per ID
    final LdMap<LdMap<dynamic>> _maps = {};

    /// Comptador de referències per cada mapa
    final LdMap<int> _refCounts = {};

    /// Informació addicional per cada mapa
    final LdMap<String> _mapTypes = {};

    // Constructors -----
    /// Constructor privat
    MapsService._() {
        Debug.info("MapsService: Inicialitzant gestor de mapes");
    }

    // Registre, Consulta i Liberació ---
    /// Registra un nou mapa i retorna el seu ID
    /// Si ja existeix un mapa idèntic, retorna l'ID d'aquest
    ///
    /// [pMap] - El mapa a registrar
    /// [pType] - Tipus del mapa (e.g., "widget", "entity", "config")
    /// [pIdent] - Identificador opcional per ajudar a generar l'ID
    String registerMap(
        LdMap<dynamic> pMap, {
        String pType = "generic",
        String? pIdent,
    }) {
        // Generar un ID per al mapa
        final String mapId = pIdent != null
            ? "${pType}_${pIdent}_${DateTime.now().millisecondsSinceEpoch}"
            : "${pType}_${DateTime.now().millisecondsSinceEpoch}_${_maps.length}";

        // Comprovar si ja existeix un mapa idèntic del mateix tipus
        for (var entry in _maps.entries) {
            if (_mapTypes[entry.key] == pType && _areMapEqual(entry.value, pMap)) {
                // Incrementar el comptador de referències
                _refCounts[entry.key] = (_refCounts[entry.key] ?? 0) + 1;
                Debug.info("MapsService: Reutilitzant mapa existent (ID: ${entry.key},
                    Tipus: $pType, Refs: ${_refCounts[entry.key]})");
                return entry.key;
            }
        }
    }
}

```

```

    // Si no existeix, guardar el mapa
    _maps[mapId] = Map<String, dynamic>.from(pMap);
    _refCounts[mapId] = 1;
    _mapTypes[mapId] = pType;

    Debug.info("MapsService: Nou mapa registrat (ID: $mapId, Tipus: $pType)");
    return mapId;
}

/// Obté el mapa per un ID
LdMap<dynamic> getMap(String pMapId) {
    assert(_maps.containsKey(pMapId), "Mapa no trobat: $pMapId");
    return _maps[pMapId]!;
}

/// Allibera una referència a un mapa
/// Si no hi ha més referències, elimina el mapa
void releaseMap(String mapId) {
    if (!_maps.containsKey(mapId)) {
        Debug.warn("MapsService: Intent d'alliberar un mapa inexistent: $mapId");
        return;
    }

    // Decrementar el comptador de referències
    _refCounts[mapId] = (_refCounts[mapId] ?? 1) - 1;

    // Si no hi ha més referències, eliminar el mapa
    if (_refCounts[mapId]! <= 0) {
        _maps.remove(mapId);
        _refCounts.remove(mapId);
        _mapTypes.remove(mapId);
        Debug.info("MapsService: Mapa eliminat (ID: $mapId)");
    } else {
        Debug.info("MapsService: Mapa alliberat (ID: $mapId, Refs restants: $
{_refCounts[mapId]})");
    }
}

// Funcionalitat interna -----
/// Comprova si dos mapes són iguals (recursivament)
bool _areMapEqual(Map<String, dynamic> pMap1, Map<String, dynamic> pMap2) {
    if (pMap1.length != pMap2.length) return false;

    for (final key in pMap1.keys) {
        if (!pMap2.containsKey(key)) return false;

        final value1 = pMap1[key];
        final value2 = pMap2[key];

        // Ignorar callbacks (no es poden comparar)

```



```

        if (value1 is Function && value2 is Function) {
            continue;
        }

        if (value1 is Map && value2 is Map) {
            if (!_areMapEqual(value1 as Map<String, dynamic>, value2 as Map<String,
dynamic>)) {
                return false;
            }
        } else if (value1 is List && value2 is List) {
            if (!_areListEqual(value1, value2)) {
                return false;
            }
        } else if (value1 != value2) {
            return false;
        }
    }

    return true;
}

/// Comprova si dues llistes són iguals (recursivament)
bool _areListEqual(List pList1, List pList2) {
    if (pList1.length != pList2.length) return false;

    for (int i = 0; i < pList1.length; i++) {
        final value1 = pList1[i];
        final value2 = pList2[i];

        if (value1 is Map && value2 is Map) {
            if (!_areMapEqual(value1 as Map<String, dynamic>, value2 as Map<String,
dynamic>)) {
                return false;
            }
        } else if (value1 is List && value2 is List) {
            if (!_areListEqual(value1, value2)) {
                return false;
            }
        } else if (value1 != value2) {
            return false;
        }
    }

    return true;
}

// Altra funcionalitat -----
/// Obté estadístiques sobre l'ús de mapes
LdMap<dynamic> getStats() {
    final stats = <String, dynamic>{
        'totalMaps': _maps.length,

```

```

        'totalReferences': _refCounts.values.fold<int>(0, (sum, count) => sum +
count),
        'mapsByType': <String, int>{},
    };

    // Comptar mapes per tipus
    for (var mapId in _maps.keys) {
        final type = _mapTypes[mapId] ?? "unknown";
        stats['mapsByType'][type] = (stats['mapsByType'][type] ?? 0) + 1;
    }

    return stats;
}

/// Neteja mapes que ja no tenen referències
int cleanup() {
    final keysToRemove = <String>[];

    // Trobar claus per eliminar
    for (var entry in _refCounts.entries) {
        if (entry.value <= 0) {
            keysToRemove.add(entry.key);
        }
    }

    // Eliminar els mapes
    for (var key in keysToRemove) {
        _maps.remove(key);
        _refCounts.remove(key);
        _mapTypes.remove(key);
    }

    Debug.info("MapsService: Neteja completada. ${keysToRemove.length} mapes
eliminats.");

    return keysToRemove.length;
}

/// Crea una còpia d'un mapa existent
String duplicateMap(String pMapId, {String? pNewIdent}) {
    assert(_maps.containsKey(pMapId), "Mapa no trobat: $pMapId");

    final originalMap = _maps[pMapId]!;
    final type = _mapTypes[pMapId] ?? "generic";

    // Registrar una còpia del mapa
    return registerMap(
        Map<String, dynamic>.from(originalMap),
        pType: type,
        pIdent: pNewIdent,
    );
}

```

```

    }
}

/// Modifica un mapa existent
/// Retorna el nou ID si es crea un nou mapa (perquè hi ha més referències)
String updateMap(String pMapId, LdMap<dynamic> pUpdates) {
    assert(_maps.containsKey(pMapId), "Mapa no trobat: $pMapId");

    // Si hi ha més d'una referència, hem de crear un nou mapa
    if (_refCounts[pMapId]! > 1) {
        // Reduir la referència al mapa original
        _refCounts[pMapId] = _refCounts[pMapId]! - 1;

        // Crear una còpia i aplicar les actualitzacions
        final originalMap = Map<String, dynamic>.from(_maps[pMapId]!);
        final type = _mapTypes[pMapId] ?? "generic";

        // Aplicar les actualitzacions
        originalMap.addAll(pUpdates);

        // Registrar el nou mapa
        return registerMap(
            originalMap,
            pType: type,
        );
    } else {
        // Si només hi ha una referència, podem modificar el mapa directament
        _maps[pMapId]!.addAll(pUpdates);
        return pMapId;
    }
}

/// Compara dos mapes i retorna les diferències
LdMap<dynamic> getDifferences(String mapId1, String mapId2) {
    assert(_maps.containsKey(mapId1), "Primer mapa no trobat: $mapId1");
    assert(_maps.containsKey(mapId2), "Segon mapa no trobat: $mapId2");

    final map1 = _maps[mapId1]!;
    final map2 = _maps[mapId2]!;
    LdMap<dynamic> differences = {};

    // Buscar diferències en map1
    for (var entry in map1.entries) {
        if (!map2.containsKey(entry.key) || map2[entry.key] != entry.value) {
            differences[entry.key] = {
                'value1': entry.value,
                'value2': map2[entry.key],
            };
        }
    }
}

```

```
// Buscar claus que només existeixen en map2
for (var key in map2.keys) {
  if (!map1.containsKey(key)) {
    differences[key] = {
      'value1': null,
      'value2': map2[key],
    };
  }
}
return differences;
}
```

Els canvis realitzats:

1. **Canvi de nom:** `ModelService` → `MapsService` per reflectir millor la seva funcionalitat
2. **Millors missatges de debug:** Tots els missatges ara diuen "MapsService" en lloc de "MapManager"
3. **Mètode nou `getDifferences`:** Permet comparar dos mapes i veure quines són les diferències
4. **Documentació millorada:** Explicacions més clares de cada mètode
5. **Import actualitzat:** S'assegura que les referències són correctes

Aquest servei és essencial per a la nova arquitectura ja que gestiona de manera eficient els mapes de configuració dels widgets i pàgines, evitant duplicats i optimitzant l'ús de memòria.

Comprova que aquesta implementació funcioni correctament abans de continuar amb el següent fitxer.

Retry

JI

Actualitzat i comprovat. Continuem.

Edit

Perfecte! Ara passem a actualitzar un widget específic per veure com quedarà amb la nova arquitectura. Començarem amb `LdTextField` que ja havíem començat a refactoritzar:

dart

```
// lib/ui/widgets/ld_text_field/ld_text_field.dart
// Widget per a l'edició de text amb suport per a internacionalització.
// Created: 2025/05/06 dt. CLA
// Updated: 2025/05/11 ds. CLA - Adaptació completa a la nova arquitectura
```

```

import 'package:flutter/material.dart';
import 'package:ld_wbench5/core/ld_widget/ld_widget_abs.dart';
import 'package:ld_wbench5/core/map_fields.dart';
import 'package:ld_wbench5/ui/widgets/ld_text_field/ld_text_field_ctrl.dart';
import 'package:ld_wbench5/ui/widgets/ld_text_field/ld_text_field_model.dart';
import 'package:ld_wbench5/utils/debug.dart';
import 'package:ld_wbench5/ui/extensions/map_extensions.dart';

export 'ld_text_field_ctrl.dart';
export 'ld_text_field_model.dart';

/// Widget per a l'edició de text amb suport per a internacionalització
class LdTextField extends LdWidgetAbs {
  /// Constructor principal
  LdTextField({
    super.key,
    String? pTag,
    String initialText = "",
    String? label,
    String? helpText,
    String? errorMessage,
    bool hasError = false,
    bool allowNull = true,
    Function(String)? onTextChanged,
    bool canFocus = true,
    bool isEnabled = true,
    bool isVisible = true,
  }) : super(config: {
    // Identificació
    cfTag: pTag ?? "LdTextField_${DateTime.now().millisecondsSinceEpoch}",

    // Propietats del MODEL (mf) - dades reals
    mfInitialText: initialText,
    mfLabel: label,
    mfHelpText: helpText,
    mfErrorMessage: errorMessage,
    mfHasError: hasError,
    mfAllowNull: allowNull,

    // Propietats del CONTROLADOR (cf) - comportament
    cfOnTextChanged: onTextChanged,
    cfCanFocus: canFocus,
    cfIsEnabled: isEnabled,
    cfIsVisible: isVisible,
  }) {
    Debug.info("$tag: LdTextField creat amb configuració");
  }

  /// Constructor alternatiu a partir d'un mapa
  LdTextField.fromMap(LdMap<dynamic> configMap, { super.key })

```

```
        : super(config: configMap) {  
            Debug.info("$tag: LdTextField creat des d'un mapa");  
        }
```

```
    @override  
    LdWidgetCtrlAbs createController() {  
        return LdTextFieldCtrl(this);  
    }
```

```
    // ACCESSORS PER A COMPATIBILITAT =====
```

```
    /// Retorna el model tipat  
    LdTextFieldModel? get model {  
        final ctrl = wCtrl;  
        if (ctrl is LdTextFieldCtrl) {  
            return ctrl.model as LdTextFieldModel?;  
        }  
        return null;  
    }
```

```
    /// Retorna el controlador tipat  
    LdTextFieldCtrl? get controller {  
        final ctrl = wCtrl;  
        if (ctrl is LdTextFieldCtrl) {  
            return ctrl;  
        }  
        return null;  
    }
```

```
    // PROPIETATS DEL MODEL =====
```

```
    /// Retorna el text actual  
    String get text => model?.text ?? "";
```

```
    /// Estableix el text  
    set text(String value) {  
        model?.updateField(mfText, value);  
    }
```

```
    /// Retorna l'etiqueta  
    String? get label => model?.label;
```

```
    /// Estableix l'etiqueta  
    set label(String? value) {  
        model?.updateField(mfLabel, value);  
    }
```

```
    /// Retorna el text d'ajuda  
    String? get helpText => model?.helpText;
```

```
    /// Estableix el text d'ajuda  
    set helpText(String? value) {  
        model?.updateField(mfHelpText, value);  
    }
```

```

    }

    /// Retorna si té error
    bool get hasError => model?.hasError ?? false;

    /// Estableix l'estat d'error
    set hasError(bool value) {
        model?.updateField(mfHasError, value);
    }

    /// Retorna el missatge d'error
    String? get errorMessage => model?.errorMessage;

    /// Estableix el missatge d'error
    set errorMessage(String? value) {
        model?.updateField(mfErrorMessage, value);
    }

    // MÈTODES DEL CONTROLADOR =====
    /// Afegeix text al final
    void appendText(String suffix) {
        controller?.addText(suffix);
    }

    /// Afegeix text al principi
    void prependText(String prefix) {
        controller?.prependText(prefix);
    }

    /// Neteja el text
    void clearText() {
        controller?.clearText();
    }

    /// Demana el focus
    void requestFocus() {
        controller?.requestFocus();
    }

    /// Allibera el focus
    void unfocus() {
        controller?.focusNode.unfocus();
    }
}

```

Els canvis principals són:

1. **Constructor simplificat**: Ara només accepta un mapa de configuració
2. **Mètode `createController`**: Implementat per crear el controlador específic

3. **Accessors tipats:** Per obtenir el model i controlador amb el tipus correcte
4. **Propietats delegades:** Totes les propietats del model són accessibles a través del widget
5. **Mètodes del controlador:** Exposats de manera senzilla a través del widget
6. **Ús de `updateField`:** Per actualitzar camps específics del model

Ara actualitzem el controlador `LdTextFieldCtrl`:

dart

```
// lib/ui/widgets/ld_text_field/ld_text_field_ctrl.dart
// Controlador del widget 'LdTextField'.
// Created: 2025/05/06 dt. CLA
// Updated: 2025/05/11 ds. CLA - Adaptació completa a la nova arquitectura

import 'package:flutter/material.dart';
import 'package:ld_wbench5/core/event_bus/ld_event.dart';
import 'package:ld_wbench5/core/ld_model_abs.dart';
import 'package:ld_wbench5/core/ld_widget/ld_widget_ctrl_abs.dart';
import 'package:ld_wbench5/core/map_fields.dart';
import 'package:ld_wbench5/ui/widgets/ld_text_field/ld_text_field.dart';
import 'package:ld_wbench5/ui/widgets/ld_text_field/ld_text_field_model.dart';
import 'package:ld_wbench5/utils/debug.dart';

/// Controlador del widget LdTextField
class LdTextFieldCtrl extends LdWidgetCtrlAbs<LdTextField> {
  // Controller per al TextField
  late final TextEditingController _textController;

  // Flag para evitar actualizaciones circulares
  bool _isUpdatingFromModel = false;

  // Constructor
  LdTextFieldCtrl(super.pWidget);

  @override
  void _createModelIfNeeded() {
    // Crear el model amb la configuració del widget
    model = LdTextFieldModel.fromMap(widget.config);

    // Inicialitzar el controller del text amb el valor del model
    _textController = TextEditingController(
      text: (model as LdTextFieldModel).text
    );

    // Configurar listener per mantenir sincronitzat el model amb el
    textController
    _textController.addListener(_onTextChange);
  }

  @override
```



```

void initialize() {
    Debug.info("$tag: Inicialitzant controlador");
}

/// S'executa quan canvia el text al textController
void _onTextChange() {
    // Evitar actualitzacions circulars
    if (_isUpdatingFromModel) return;

    final text = _textController.text;

    if (text != (model as LdTextFieldModel).text) {
        Debug.info("$tag: Text canviat des del teclat a '$text'");
        // Actualitzar el model amb el nou text
        (model as LdTextFieldModel).updateField(mfText, text);
    }
}

/// Actualitza el text del controller a partir del model
void _updateControllerText() {
    final modelText = (model as LdTextFieldModel).text;
    if (_textController.text != modelText) {
        _isUpdatingFromModel = true;

        _textController.text = modelText;
        // Posicionar el cursor al final
        _textController.selection = TextSelection.fromPosition(
            TextPosition(offset: _textController.text.length),
        );

        _isUpdatingFromModel = false;
        Debug.info("$tag: TextController actualitzat a '$modelText'");
    }
}

/// Nou mètode per afegir text directament
void addText(String text) {
    Debug.info("$tag: Afegint text '$text' directament des del controlador");

    final model = this.model as LdTextFieldModel;
    final newText = model.text + text;
    model.updateField(mfText, newText);
}

/// Nou mètode per afegir text al principi
void prependText(String prefix) {
    Debug.info("$tag: Afegint prefix '$prefix' directament des del controlador");

    final model = this.model as LdTextFieldModel;

```

```

        final newText = prefix + model.text;
        model.updateField(mfText, newText);
    }

    /// Nou mètode per netejar el text
    void clearText() {
        Debug.info("$tag: Netejant text des del controlador");

        final model = this.model as LdTextFieldModel;
        model.updateField(mfText, "");
    }

    @override
    void update() {
        // Actualitzar el text del controller amb el del model
        _updateControllerText();
    }

    @override
    void onEvent(LdEvent event) {
        Debug.info("$tag: Rebut event ${event.eType.name}");

        // Gestionem els events que ens interessin
        if (event.eType == EventType.languageChanged) {
            Debug.info("$tag: Processant canvi d'idioma");
            if (mounted) {
                setState(() {
                    // Els texts traduïts s'actualitzaran automàticament
                });
            }
        }
    }

    @override
    void onModelChanged(LdModelAbs pModel, void Function() updateFunction) {
        Debug.info("$tag: Model ha canviat");

        // Executar la funció d'actualització
        updateFunction();

        // Actualitzar el TextEditingController si és necessari
        _updateControllerText();

        // Reconstruir si està muntat
        if (mounted) {
            setState(() {
                Debug.info("$tag: Reconstruint després del canvi del model");
            });
        }
    }
}

```

```

@override
void dispose() {
  _textController.removeListener(_onTextChange);
  _textController.dispose();
  super.dispose();
}

@override
Widget buildContent(BuildContext context) {
  final model = this.model as LdTextFieldModel;
  final theme = Theme.of(context);

  return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    mainAxisAlignment: MainAxisAlignment.min,
    children: [
      // El TextField amb l'etiqueta
      TextField(
        controller: _textController,
        focusNode: focusNode,
        enabled: isEnabled,
        decoration: InputDecoration(
          labelText: model.label,
          helperText: model.helperText,
          errorText: model.hasError ? model.errorMessage : null,
          border: const OutlineInputBorder(),
          isDense: true,
        ),
        style: theme.textTheme.bodyMedium,
      ),
    ],
  );
}

```

Els canvis principals en el controlador:

1. **`_createModelIfNeeded()`** **implementat**: Crea el model des del mapa de configuració
2. **Gestió de callbacks simplificada**: El callback d'onChange es gestiona a través del model
3. **Ús de `updateField`**: Per actualitzar camps específics del model
4. **Model tipat**: Accés al model com a `LdTextFieldModel`

Comprova que aquesta implementació funcioni correctament abans de continuar amb el model.

Retry

JI

Recorda que ja no passem 'super.key' amb el constructor.

Edit

Tens raó! Ho havia passat per alt. Anem a corregir-ho. El constructor de `LdTextField` no hauria de rebre la key com a paràmetre:

dart

```
// lib/ui/widgets/ld_text_field/ld_text_field.dart
// Widget per a l'edició de text amb suport per a internacionalització.
// Created: 2025/05/06 dt. CLA
// Updated: 2025/05/11 ds. CLA - Adaptació completa a la nova arquitectura

import 'package:flutter/material.dart';
import 'package:ld_wbench5/core/ld_widget/ld_widget_abs.dart';
import 'package:ld_wbench5/core/map_fields.dart';
import 'package:ld_wbench5/ui/widgets/ld_text_field/ld_text_field_ctrl.dart';
import 'package:ld_wbench5/ui/widgets/ld_text_field/ld_text_field_model.dart';
import 'package:ld_wbench5/utils/debug.dart';
import 'package:ld_wbench5/ui/extensions/map_extensions.dart';

export 'ld_text_field_ctrl.dart';
export 'ld_text_field_model.dart';

/// Widget per a l'edició de text amb suport per a internacionalització
class LdTextField extends LdWidgetAbs {
  /// Constructor principal
  LdTextField({
    String? pTag,
    String initialText = "",
    String? label,
    String? helpText,
    String? errorMessage,
    bool hasError = false,
    bool allowNull = true,
    Function(String)? onTextChanged,
    bool canFocus = true,
    bool isEnabled = true,
    bool isVisible = true,
  }) : super(config: {
    // Identificació
    cfTag: pTag ?? "LdTextField_${DateTime.now().millisecondsSinceEpoch}",

    // Propietats del MODEL (mf) - dades reals
    mfInitialText: initialText,
    mfLabel: label,
    mfHelpText: helpText,
    mfErrorMessage: errorMessage,
    mfHasError: hasError,
    mfAllowNull: allowNull,
```

```

// Propietats del CONTROLADOR (cf) - comportament
cfOnTextChanged: onTextChanged,
cfCanFocus: canFocus,
cfIsEnabled: isEnabled,
cfIsVisible: isVisible,
}) {
    Debug.info("$tag: LdTextField creat amb configuració");
}

```

```

/// Constructor alternatiu a partir d'un mapa
LdTextField.fromMap(LdMap<dynamic> configMap)
    : super(config: configMap) {
    Debug.info("$tag: LdTextField creat des d'un mapa");
}

```

```

@override
LdWidgetCtrlAbs createController() {
    return LdTextFieldCtrl(this);
}

```

```

// ACCESSORS PER A COMPATIBILITAT =====
/// Retorna el model tipat
LdTextFieldModel? get model {
    final ctrl = wCtrl;
    if (ctrl is LdTextFieldCtrl) {
        return ctrl.model as LdTextFieldModel?;
    }
    return null;
}

```

```

/// Retorna el controlador tipat
LdTextFieldCtrl? get controller {
    final ctrl = wCtrl;
    if (ctrl is LdTextFieldCtrl) {
        return ctrl;
    }
    return null;
}

```

```

// PROPIETATS DEL MODEL =====
/// Retorna el text actual
String get text => model?.text ?? "";

/// Estableix el text
set text(String value) {
    model?.updateField(mfText, value);
}

```

```

/// Retorna l'etiqueta
String? get label => model?.label;

```

```

    /// Estableix l'etiqueta
    set label(String? value) {
        model?.updateField(mfLabel, value);
    }

    /// Retorna el text d'ajuda
    String? get helpText => model?.helpText;

    /// Estableix el text d'ajuda
    set helpText(String? value) {
        model?.updateField(mfHelpText, value);
    }

    /// Retorna si té error
    bool get hasError => model?.hasError ?? false;

    /// Estableix l'estat d'error
    set hasError(bool value) {
        model?.updateField(mfHasError, value);
    }

    /// Retorna el missatge d'error
    String? get errorMessage => model?.errorMessage;

    /// Estableix el missatge d'error
    set errorMessage(String? value) {
        model?.updateField(mfErrorMessage, value);
    }

    // MÈTODES DEL CONTROLADOR =====
    /// Afegeix text al final
    void appendText(String suffix) {
        controller?.addText(suffix);
    }

    /// Afegeix text al principi
    void prependText(String prefix) {
        controller?.prependText(prefix);
    }

    /// Neteja el text
    void clearText() {
        controller?.clearText();
    }

    /// Demana el focus
    void requestFocus() {
        controller?.requestFocus();
    }

```

```
/// Allibera el focus
void unfocus() {
  controller?.focusNode.unfocus();
}
}
```

Ara ja està correcte. He eliminat:

- `super.key` del constructor principal
- El paràmetre `{ super.key }` del constructor `fromMap`

L'arquitectura optimitzada ja no necessita que els widgets rebin la key com a paràmetre extern, ja que la `GlobalKey` es gestiona internament a través del mixin `LdTaggableMixin`.

Aquesta és la implementació correcta del `LdTextField` seguint la nova arquitectura. Pots comprovar que funcioni abans de continuar amb el model.

Retry

JI

Classe actualitzada i validada. Continuem.

Edit

Excellent! Ara actualitzem el model `LdTextFieldModel`:

dart

```
// lib/ui/widgets/ld_text_field/ld_text_field_model.dart
// Model de dades del widget 'LdTextField'.
// Created: 2025/05/06 dt. CLA
// Updated: 2025/05/11 ds. CLA - Adaptació completa a la nova arquitectura
```

```
import 'package:ld_wbench5/core/ld_widget/ld_widget_model_abs.dart';
import 'package:ld_wbench5/core/map_fields.dart';
import 'package:ld_wbench5/ui/widgets/ld_text_field/ld_text_field.dart';
import 'package:ld_wbench5/utils/debug.dart';
import 'package:ld_wbench5/ui/extensions/map_extensions.dart';
import 'package:ld_wbench5/utils/str_full_set.dart';
```

```
/// Model de dades del widget LdTextField
class LdTextFieldModel extends LdWidgetModelAbs<LdTextField> {
  // Text intern
  String _text = "";
  String get text => _text;
  set text(String value) {
    if (_text != value) {
      notifyListeners() {
        _text = value;
      }
    }
  }
}
```

```
Debug.info("$tag: Text canviat a '$value'");
```

```
// Si tenim un handler extern, notificar-lo
```

```
if (_onTextChanged != null) {
```

```
    _onTextChanged!(_text);
```

```
}
```

```
});
```

```
}
```

```
}
```

```
// Etiqueta
```

```
final StrFullSet _label = StrFullSet();
```

```
String? get label => _label.tx;
```

```
set label(String? value) {
```

```
    notifyListeners(() {
```

```
        _label.t = value;
```

```
        Debug.info("$tag: Etiqueta canviada a '$value'");
```

```
    });
```

```
}
```

```
// Text d'ajuda
```

```
final StrFullSet _helpText = StrFullSet();
```

```
String? get helpText => _helpText.tx;
```

```
set helpText(String? value) {
```

```
    notifyListeners(() {
```

```
        _helpText.t = value;
```

```
        Debug.info("$tag: Text d'ajuda canviat a '$value'");
```

```
    });
```

```
}
```

```
// Estat d'error
```

```
bool _hasError = false;
```

```
bool get hasError => _hasError;
```

```
set hasError(bool value) {
```

```
    if (_hasError != value) {
```

```
        notifyListeners(() {
```

```
            _hasError = value;
```

```
            Debug.info("$tag: Estat d'error canviat a '$value'");
```

```
        });
```

```
}
```

```
}
```

```
// Missatge d'error
```

```
final StrFullSet _errorMessage = StrFullSet();
```

```
String? get errorMessage => _errorMessage.tx;
```

```
set errorMessage(String? value) {
```

```
    notifyListeners(() {
```

```
        _errorMessage.t = value;
```

```
        Debug.info("$tag: Missatge d'error canviat a '$value'");
```

```
    });
```



```

    }
}

// Pot ser nul·la
bool _allowNull = true;
bool get allowNull => _allowNull;
set allowNull(bool value) {
    if (_allowNull != value) {
        notifyListeners() {
            _allowNull = value;
            Debug.info("$tag: Pot ser nul·la canviat a $value");
        });
    }
}

// Callback quan canvia el text
Function(String)? _onTextChanged;
Function(String)? get onTextChanged => _onTextChanged;
set onTextChanged(Function(String)? value) {
    _onTextChanged = value;
}

// Constructor des d'un mapa
LdTextFieldModel.fromMap(LdMap<dynamic> pMap) : super.fromMap(pMap) {
    // Les propietats ja es carregaran a fromMap
}

// Mapeig
@override
void fromMap(LdMap<dynamic> pMap) {
    super.fromMap(pMap);

    // Carregar propietats del model (mf)
    _text = pMap[mfInitialText] as String? ?? "";
    _label.t = pMap[mfLabel] as String?;
    _helpText.t = pMap[mfHelpText] as String?;
    _errorMessage.t = pMap[mfErrorMessage] as String?;
    _hasError = pMap[mfHasError] as bool? ?? false;
    _allowNull = pMap[mfAllowNull] as bool? ?? true;

    // Carregar callback si existeix (cf)
    _onTextChanged = pMap[cfOnTextChanged] as Function(String)?;

    Debug.info("$tag: Model carregat des de mapa");
}

@override
LdMap<dynamic> toMap() {
    LdMap<dynamic> map = super.toMap();
    map.addAll({
        mfText: _text,

```

```

        mfLabel: _label.t,
        mfHelpText: _helpText.t,
        mfErrorMessage: _errorMessage.t,
        mfHasError: _hasError,
        mfAllowNull: _allowNull,
        // No guardem el callback al mapa ja que no és serialitzable
    });
    return map;
}

@override
dynamic getField({required String pKey, bool pCouldBeNull = true, String?
pErrorMsg}) {
    switch (pKey) {
        case mfText: return text;
        case mfLabel: return label;
        case mfHelpText: return helpText;
        case mfErrorMessage: return errorMessage;
        case mfHasError: return hasError;
        case mfAllowNull: return allowNull;
        default: return super.getField(
            pKey: pKey,
            pCouldBeNull: pCouldBeNull,
            pErrorMsg: pErrorMsg
        );
    }
}

@override
void setField({required String pKey, dynamic pValue, bool pCouldBeNull =
true, String? pErrorMsg}) {
    switch (pKey) {
        case mfText:
            if (pValue is String) text = pValue;
            break;
        case mfLabel:
            if (pValue is String || pValue == null) label = pValue;
            break;
        case mfHelpText:
            if (pValue is String || pValue == null) helpText = pValue;
            break;
        case mfErrorMessage:
            if (pValue is String || pValue == null) errorMessage = pValue;
            break;
        case mfHasError:
            if (pValue is bool) hasError = pValue;
            break;
        case mfAllowNull:
            if (pValue is bool) allowNull = pValue;
            break;
    }
}

```

```

        default:
            super.setField(
                pKey: pKey,
                pValue: pValue,
                pCouldBeNull: pCouldBeNull,
                pErrorMsg: pErrorMsg
            );
        }
    }

    // Validació
    bool validate() {
        if (!allowNull && _text.isEmpty) {
            hasError = true;
            errorMessage = "Aquest camp és obligatori";
            return false;
        }

        // Neteja l'error si la validació passa
        hasError = false;
        errorMessage = null;

        return true;
    }

    // Netejar el text i l'estat d'error
    void clear() {
        notifyListeners(() {
            _text = "";
            _hasError = false;
            _errorMessage.t = null;
            Debug.info("$tag: Model netejat");
        });
    }
}

```

Els canvis principals en el model:

1. **Constructor fromMap**: Hereta de `LdWidgetModelAbs.fromMap`
2. **Propietats amb getters/setters**: Cada propietat notifica als observadors quan canvia
3. **Suport complet per camps mf**: Totes les constants `mf*` tenen suport complet
4. **Suport per callback**: El callback `onTextChanged` es carrega del mapa però no es serialitza
5. **Validació integrada**: Mètode `validate()` per comprovar el contingut
6. **Mètode clear()**: Per netejar el contingut i l'estat d'error
7. **Ús de StrFullSet**: Per a camps que poden tenir traduccions

Aquesta implementació fa que el model sigui completament compatible amb el sistema de mapes i
proporcioni totes les funcionalitats necessàries per gestionar un camp de text.

Comprova que aquesta implementació funcioni correctament abans de continuar amb el següent
widget.

Retry

[Claude can make mistakes. Please double-check responses.](#)