//

// Generated code. Do not modify.

// source: diff.proto

//

// @dart = 2.12

// ignore\_for\_file: annotate\_overrides, camel\_case\_types, comment\_references

// ignore\_for\_file: constant\_identifier\_names, library\_prefixes

// ignore\_for\_file: non\_constant\_identifier\_names, prefer\_final\_fields

// ignore\_for\_file: unnecessary\_import, unnecessary\_this, unused\_import

import 'dart:core' as $core;

import 'package:protobuf/protobuf.dart' as $pb;

class DiffRequest extends $pb.GeneratedMessage {

factory DiffRequest({

$core.String? text1,

$core.String? text2,

}) {

final $result = create();

if (text1 != null) {

$result.text1 = text1;

}

if (text2 != null) {

$result.text2 = text2;

}

return $result;

}

DiffRequest.\_() : super();

factory DiffRequest.fromBuffer($core.List<$core.int> i, [$pb.ExtensionRegistry r = $pb.ExtensionRegistry.EMPTY]) => create()..mergeFromBuffer(i, r);

factory DiffRequest.fromJson($core.String i, [$pb.ExtensionRegistry r = $pb.ExtensionRegistry.EMPTY]) => create()..mergeFromJson(i, r);

static final $pb.BuilderInfo \_i = $pb.BuilderInfo(\_omitMessageNames ? '' : 'DiffRequest', package: const $pb.PackageName(\_omitMessageNames ? '' : 'diff'), createEmptyInstance: create)

..aOS(1, \_omitFieldNames ? '' : 'text1')

..aOS(2, \_omitFieldNames ? '' : 'text2')

..hasRequiredFields = false

;

@$core.Deprecated(

'Using this can add significant overhead to your binary. '

'Use [GeneratedMessageGenericExtensions.deepCopy] instead. '

'Will be removed in next major version')

DiffRequest clone() => DiffRequest()..mergeFromMessage(this);

@$core.Deprecated(

'Using this can add significant overhead to your binary. '

'Use [GeneratedMessageGenericExtensions.rebuild] instead. '

'Will be removed in next major version')

DiffRequest copyWith(void Function(DiffRequest) updates) => super.copyWith((message) => updates(message as DiffRequest)) as DiffRequest;

$pb.BuilderInfo get info\_ => \_i;

@$core.pragma('dart2js:noInline')

static DiffRequest create() => DiffRequest.\_();

DiffRequest createEmptyInstance() => create();

static $pb.PbList<DiffRequest> createRepeated() => $pb.PbList<DiffRequest>();

@$core.pragma('dart2js:noInline')

static DiffRequest getDefault() => \_defaultInstance ??= $pb.GeneratedMessage.$\_defaultFor<DiffRequest>(create);

static DiffRequest? \_defaultInstance;

@$pb.TagNumber(1)

$core.String get text1 => $\_getSZ(0);

@$pb.TagNumber(1)

set text1($core.String v) { $\_setString(0, v); }

@$pb.TagNumber(1)

$core.bool hasText1() => $\_has(0);

@$pb.TagNumber(1)

void clearText1() => clearField(1);

@$pb.TagNumber(2)

$core.String get text2 => $\_getSZ(1);

@$pb.TagNumber(2)

set text2($core.String v) { $\_setString(1, v); }

@$pb.TagNumber(2)

$core.bool hasText2() => $\_has(1);

@$pb.TagNumber(2)

void clearText2() => clearField(2);

}

class DiffResponse extends $pb.GeneratedMessage {

factory DiffResponse({

$core.String? message,

$core.double? similarity,

}) {

final $result = create();

if (message != null) {

$result.message = message;

}

if (similarity != null) {

$result.similarity = similarity;

}

return $result;

}

DiffResponse.\_() : super();

factory DiffResponse.fromBuffer($core.List<$core.int> i, [$pb.ExtensionRegistry r = $pb.ExtensionRegistry.EMPTY]) => create()..mergeFromBuffer(i, r);

factory DiffResponse.fromJson($core.String i, [$pb.ExtensionRegistry r = $pb.ExtensionRegistry.EMPTY]) => create()..mergeFromJson(i, r);

static final $pb.BuilderInfo \_i = $pb.BuilderInfo(\_omitMessageNames ? '' : 'DiffResponse', package: const $pb.PackageName(\_omitMessageNames ? '' : 'diff'), createEmptyInstance: create)

..aOS(1, \_omitFieldNames ? '' : 'message')

..a<$core.double>(2, \_omitFieldNames ? '' : 'similarity', $pb.PbFieldType.OD)

..hasRequiredFields = false

;

@$core.Deprecated(

'Using this can add significant overhead to your binary. '

'Use [GeneratedMessageGenericExtensions.deepCopy] instead. '

'Will be removed in next major version')

DiffResponse clone() => DiffResponse()..mergeFromMessage(this);

@$core.Deprecated(

'Using this can add significant overhead to your binary. '

'Use [GeneratedMessageGenericExtensions.rebuild] instead. '

'Will be removed in next major version')

DiffResponse copyWith(void Function(DiffResponse) updates) => super.copyWith((message) => updates(message as DiffResponse)) as DiffResponse;

$pb.BuilderInfo get info\_ => \_i;

@$core.pragma('dart2js:noInline')

static DiffResponse create() => DiffResponse.\_();

DiffResponse createEmptyInstance() => create();

static $pb.PbList<DiffResponse> createRepeated() => $pb.PbList<DiffResponse>();

@$core.pragma('dart2js:noInline')

static DiffResponse getDefault() => \_defaultInstance ??= $pb.GeneratedMessage.$\_defaultFor<DiffResponse>(create);

static DiffResponse? \_defaultInstance;

@$pb.TagNumber(1)

$core.String get message => $\_getSZ(0);

@$pb.TagNumber(1)

set message($core.String v) { $\_setString(0, v); }

@$pb.TagNumber(1)

$core.bool hasMessage() => $\_has(0);

@$pb.TagNumber(1)

void clearMessage() => clearField(1);

@$pb.TagNumber(2)

$core.double get similarity => $\_getN(1);

@$pb.TagNumber(2)

set similarity($core.double v) { $\_setDouble(1, v); }

@$pb.TagNumber(2)

$core.bool hasSimilarity() => $\_has(1);

@$pb.TagNumber(2)

void clearSimilarity() => clearField(2);

}

const \_omitFieldNames = $core.bool.fromEnvironment('protobuf.omit\_field\_names');

const \_omitMessageNames = $core.bool.fromEnvironment('protobuf.omit\_message\_names');

//

// Generated code. Do not modify.

// source: diff.proto

//

// @dart = 2.12

// ignore\_for\_file: annotate\_overrides, camel\_case\_types, comment\_references

// ignore\_for\_file: constant\_identifier\_names, library\_prefixes

// ignore\_for\_file: non\_constant\_identifier\_names, prefer\_final\_fields

// ignore\_for\_file: unnecessary\_import, unnecessary\_this, unused\_import

import 'dart:async' as $async;

import 'dart:core' as $core;

import 'package:grpc/service\_api.dart' as $grpc;

import 'package:protobuf/protobuf.dart' as $pb;

import 'diff.pb.dart' as $0;

export 'diff.pb.dart';

@$pb.GrpcServiceName('diff.Diff')

class DiffClient extends $grpc.Client {

static final \_$getResult = $grpc.ClientMethod<$0.DiffRequest, $0.DiffResponse>(

'/diff.Diff/GetResult',

($0.DiffRequest value) => value.writeToBuffer(),

($core.List<$core.int> value) => $0.DiffResponse.fromBuffer(value));

DiffClient($grpc.ClientChannel channel,

{$grpc.CallOptions? options,

$core.Iterable<$grpc.ClientInterceptor>? interceptors})

: super(channel, options: options,

interceptors: interceptors);

$grpc.ResponseFuture<$0.DiffResponse> getResult($0.DiffRequest request, {$grpc.CallOptions? options}) {

return $createUnaryCall(\_$getResult, request, options: options);

}

}

@$pb.GrpcServiceName('diff.Diff')

abstract class DiffServiceBase extends $grpc.Service {

$core.String get $name => 'diff.Diff';

DiffServiceBase() {

$addMethod($grpc.ServiceMethod<$0.DiffRequest, $0.DiffResponse>(

'GetResult',

getResult\_Pre,

false,

false,

($core.List<$core.int> value) => $0.DiffRequest.fromBuffer(value),

($0.DiffResponse value) => value.writeToBuffer()));

}

$async.Future<$0.DiffResponse> getResult\_Pre($grpc.ServiceCall call, $async.Future<$0.DiffRequest> request) async {

return getResult(call, await request);

}

$async.Future<$0.DiffResponse> getResult($grpc.ServiceCall call, $0.DiffRequest request);

}

//

//  Generated code. Do not modify.

//  source: diff.proto

//

// @dart = 2.12

// ignore\_for\_file: annotate\_overrides, camel\_case\_types, comment\_references

// ignore\_for\_file: constant\_identifier\_names, library\_prefixes

// ignore\_for\_file: non\_constant\_identifier\_names, prefer\_final\_fields

// ignore\_for\_file: unnecessary\_import, unnecessary\_this, unused\_import

import 'dart:convert' as $convert;

import 'dart:core' as $core;

import 'dart:typed\_data' as $typed\_data;

@$core.Deprecated('Use diffRequestDescriptor instead')

const DiffRequest$json = {

  '1': 'DiffRequest',

  '2': [

    {'1': 'text1', '3': 1, '4': 1, '5': 9, '10': 'text1'},

    {'1': 'text2', '3': 2, '4': 1, '5': 9, '10': 'text2'},

  ],

};

/// Descriptor for `DiffRequest`. Decode as a `google.protobuf.DescriptorProto`.

final $typed\_data.Uint8List diffRequestDescriptor = $convert.base64Decode(

    'CgtEaWZmUmVxdWVzdBIUCgV0ZXh0MRgBIAEoCVIFdGV4dDESFAoFdGV4dDIYAiABKAlSBXRleH'

    'Qy');

@$core.Deprecated('Use diffResponseDescriptor instead')

const DiffResponse$json = {

  '1': 'DiffResponse',

  '2': [

    {'1': 'message', '3': 1, '4': 1, '5': 9, '10': 'message'},

    {'1': 'similarity', '3': 2, '4': 1, '5': 1, '10': 'similarity'},

  ],

};

/// Descriptor for `DiffResponse`. Decode as a `google.protobuf.DescriptorProto`.

final $typed\_data.Uint8List diffResponseDescriptor = $convert.base64Decode(

    'CgxEaWZmUmVzcG9uc2USGAoHbWVzc2FnZRgBIAEoCVIHbWVzc2FnZRIeCgpzaW1pbGFyaXR5GA'

    'IgASgBUgpzaW1pbGFyaXR5');

//

//  Generated code. Do not modify.

//  source: nlp.proto

//

// @dart = 2.12

// ignore\_for\_file: annotate\_overrides, camel\_case\_types, comment\_references

// ignore\_for\_file: constant\_identifier\_names, library\_prefixes

// ignore\_for\_file: non\_constant\_identifier\_names, prefer\_final\_fields

// ignore\_for\_file: unnecessary\_import, unnecessary\_this, unused\_import

import 'dart:core' as $core;

import 'package:protobuf/protobuf.dart' as $pb;

class NlpRequest extends $pb.GeneratedMessage {

  factory NlpRequest({

    $core.String? message,

  }) {

    final $result = create();

    if (message != null) {

      $result.message = message;

    }

    return $result;

  }

  NlpRequest.\_() : super();

  factory NlpRequest.fromBuffer($core.List<$core.int> i, [$pb.ExtensionRegistry r = $pb.ExtensionRegistry.EMPTY]) => create()..mergeFromBuffer(i, r);

  factory NlpRequest.fromJson($core.String i, [$pb.ExtensionRegistry r = $pb.ExtensionRegistry.EMPTY]) => create()..mergeFromJson(i, r);

  static final $pb.BuilderInfo \_i = $pb.BuilderInfo(\_omitMessageNames ? '' : 'NlpRequest', package: const $pb.PackageName(\_omitMessageNames ? '' : 'nlp'), createEmptyInstance: create)

    ..aOS(1, \_omitFieldNames ? '' : 'message')

    ..hasRequiredFields = false

  ;

  @$core.Deprecated(

  'Using this can add significant overhead to your binary. '

  'Use [GeneratedMessageGenericExtensions.deepCopy] instead. '

  'Will be removed in next major version')

  NlpRequest clone() => NlpRequest()..mergeFromMessage(this);

  @$core.Deprecated(

  'Using this can add significant overhead to your binary. '

  'Use [GeneratedMessageGenericExtensions.rebuild] instead. '

  'Will be removed in next major version')

  NlpRequest copyWith(void Function(NlpRequest) updates) => super.copyWith((message) => updates(message as NlpRequest)) as NlpRequest;

  $pb.BuilderInfo get info\_ => \_i;

  @$core.pragma('dart2js:noInline')

  static NlpRequest create() => NlpRequest.\_();

  NlpRequest createEmptyInstance() => create();

  static $pb.PbList<NlpRequest> createRepeated() => $pb.PbList<NlpRequest>();

  @$core.pragma('dart2js:noInline')

  static NlpRequest getDefault() => \_defaultInstance ??= $pb.GeneratedMessage.$\_defaultFor<NlpRequest>(create);

  static NlpRequest? \_defaultInstance;

  @$pb.TagNumber(1)

  $core.String get message => $\_getSZ(0);

  @$pb.TagNumber(1)

  set message($core.String v) { $\_setString(0, v); }

  @$pb.TagNumber(1)

  $core.bool hasMessage() => $\_has(0);

  @$pb.TagNumber(1)

  void clearMessage() => clearField(1);

}

class NlpReply extends $pb.GeneratedMessage {

  factory NlpReply({

    $core.Iterable<Detail>? details,

  }) {

    final $result = create();

    if (details != null) {

      $result.details.addAll(details);

    }

    return $result;

  }

  NlpReply.\_() : super();

  factory NlpReply.fromBuffer($core.List<$core.int> i, [$pb.ExtensionRegistry r = $pb.ExtensionRegistry.EMPTY]) => create()..mergeFromBuffer(i, r);

  factory NlpReply.fromJson($core.String i, [$pb.ExtensionRegistry r = $pb.ExtensionRegistry.EMPTY]) => create()..mergeFromJson(i, r);

  static final $pb.BuilderInfo \_i = $pb.BuilderInfo(\_omitMessageNames ? '' : 'NlpReply', package: const $pb.PackageName(\_omitMessageNames ? '' : 'nlp'), createEmptyInstance: create)

    ..pc<Detail>(1, \_omitFieldNames ? '' : 'details', $pb.PbFieldType.PM, subBuilder: Detail.create)

    ..hasRequiredFields = false

  ;

  @$core.Deprecated(

  'Using this can add significant overhead to your binary. '

  'Use [GeneratedMessageGenericExtensions.deepCopy] instead. '

  'Will be removed in next major version')

  NlpReply clone() => NlpReply()..mergeFromMessage(this);

  @$core.Deprecated(

  'Using this can add significant overhead to your binary. '

  'Use [GeneratedMessageGenericExtensions.rebuild] instead. '

  'Will be removed in next major version')

  NlpReply copyWith(void Function(NlpReply) updates) => super.copyWith((message) => updates(message as NlpReply)) as NlpReply;

  $pb.BuilderInfo get info\_ => \_i;

  @$core.pragma('dart2js:noInline')

  static NlpReply create() => NlpReply.\_();

  NlpReply createEmptyInstance() => create();

  static $pb.PbList<NlpReply> createRepeated() => $pb.PbList<NlpReply>();

  @$core.pragma('dart2js:noInline')

  static NlpReply getDefault() => \_defaultInstance ??= $pb.GeneratedMessage.$\_defaultFor<NlpReply>(create);

  static NlpReply? \_defaultInstance;

  @$pb.TagNumber(1)

  $core.List<Detail> get details => $\_getList(0);

}

class Detail extends $pb.GeneratedMessage {

  factory Detail({

    $core.String? name,

    $core.String? type,

  }) {

    final $result = create();

    if (name != null) {

      $result.name = name;

    }

    if (type != null) {

      $result.type = type;

    }

    return $result;

  }

  Detail.\_() : super();

  factory Detail.fromBuffer($core.List<$core.int> i, [$pb.ExtensionRegistry r = $pb.ExtensionRegistry.EMPTY]) => create()..mergeFromBuffer(i, r);

  factory Detail.fromJson($core.String i, [$pb.ExtensionRegistry r = $pb.ExtensionRegistry.EMPTY]) => create()..mergeFromJson(i, r);

  static final $pb.BuilderInfo \_i = $pb.BuilderInfo(\_omitMessageNames ? '' : 'Detail', package: const $pb.PackageName(\_omitMessageNames ? '' : 'nlp'), createEmptyInstance: create)

    ..aOS(1, \_omitFieldNames ? '' : 'name')

    ..aOS(2, \_omitFieldNames ? '' : 'type')

    ..hasRequiredFields = false

  ;

  @$core.Deprecated(

  'Using this can add significant overhead to your binary. '

  'Use [GeneratedMessageGenericExtensions.deepCopy] instead. '

  'Will be removed in next major version')

  Detail clone() => Detail()..mergeFromMessage(this);

  @$core.Deprecated(

  'Using this can add significant overhead to your binary. '

  'Use [GeneratedMessageGenericExtensions.rebuild] instead. '

  'Will be removed in next major version')

  Detail copyWith(void Function(Detail) updates) => super.copyWith((message) => updates(message as Detail)) as Detail;

  $pb.BuilderInfo get info\_ => \_i;

  @$core.pragma('dart2js:noInline')

  static Detail create() => Detail.\_();

  Detail createEmptyInstance() => create();

  static $pb.PbList<Detail> createRepeated() => $pb.PbList<Detail>();

  @$core.pragma('dart2js:noInline')

  static Detail getDefault() => \_defaultInstance ??= $pb.GeneratedMessage.$\_defaultFor<Detail>(create);

  static Detail? \_defaultInstance;

  @$pb.TagNumber(1)

  $core.String get name => $\_getSZ(0);

  @$pb.TagNumber(1)

  set name($core.String v) { $\_setString(0, v); }

  @$pb.TagNumber(1)

  $core.bool hasName() => $\_has(0);

  @$pb.TagNumber(1)

  void clearName() => clearField(1);

  @$pb.TagNumber(2)

  $core.String get type => $\_getSZ(1);

  @$pb.TagNumber(2)

  set type($core.String v) { $\_setString(1, v); }

  @$pb.TagNumber(2)

  $core.bool hasType() => $\_has(1);

  @$pb.TagNumber(2)

  void clearType() => clearField(2);

}

const \_omitFieldNames = $core.bool.fromEnvironment('protobuf.omit\_field\_names');

const \_omitMessageNames = $core.bool.fromEnvironment('protobuf.omit\_message\_names');

//

// Generated code. Do not modify.

// source: nlp.proto

//

// @dart = 2.12

// ignore\_for\_file: annotate\_overrides, camel\_case\_types, comment\_references

// ignore\_for\_file: constant\_identifier\_names, library\_prefixes

// ignore\_for\_file: non\_constant\_identifier\_names, prefer\_final\_fields

// ignore\_for\_file: unnecessary\_import, unnecessary\_this, unused\_import

import 'dart:async' as $async;

import 'dart:core' as $core;

import 'package:grpc/service\_api.dart' as $grpc;

import 'package:protobuf/protobuf.dart' as $pb;

import 'nlp.pb.dart' as $0;

export 'nlp.pb.dart';

@$pb.GrpcServiceName('nlp.NlpExtractor')

class NlpExtractorClient extends $grpc.Client {

static final \_$getResult = $grpc.ClientMethod<$0.NlpRequest, $0.NlpReply>(

'/nlp.NlpExtractor/GetResult',

($0.NlpRequest value) => value.writeToBuffer(),

($core.List<$core.int> value) => $0.NlpReply.fromBuffer(value));

NlpExtractorClient($grpc.ClientChannel channel,

{$grpc.CallOptions? options,

$core.Iterable<$grpc.ClientInterceptor>? interceptors})

: super(channel, options: options,

interceptors: interceptors);

$grpc.ResponseFuture<$0.NlpReply> getResult($0.NlpRequest request, {$grpc.CallOptions? options}) {

return $createUnaryCall(\_$getResult, request, options: options);

}

}

@$pb.GrpcServiceName('nlp.NlpExtractor')

abstract class NlpExtractorServiceBase extends $grpc.Service {

$core.String get $name => 'nlp.NlpExtractor';

NlpExtractorServiceBase() {

$addMethod($grpc.ServiceMethod<$0.NlpRequest, $0.NlpReply>(

'GetResult',

getResult\_Pre,

false,

false,

($core.List<$core.int> value) => $0.NlpRequest.fromBuffer(value),

($0.NlpReply value) => value.writeToBuffer()));

}

$async.Future<$0.NlpReply> getResult\_Pre($grpc.ServiceCall call, $async.Future<$0.NlpRequest> request) async {

return getResult(call, await request);

}

$async.Future<$0.NlpReply> getResult($grpc.ServiceCall call, $0.NlpRequest request);

}

//

// Generated code. Do not modify.

// source: nlp.proto

//

// @dart = 2.12

// ignore\_for\_file: annotate\_overrides, camel\_case\_types, comment\_references

// ignore\_for\_file: constant\_identifier\_names, library\_prefixes

// ignore\_for\_file: non\_constant\_identifier\_names, prefer\_final\_fields

// ignore\_for\_file: unnecessary\_import, unnecessary\_this, unused\_import

import 'dart:convert' as $convert;

import 'dart:core' as $core;

import 'dart:typed\_data' as $typed\_data;

@$core.Deprecated('Use nlpRequestDescriptor instead')

const NlpRequest$json = {

'1': 'NlpRequest',

'2': [

{'1': 'message', '3': 1, '4': 1, '5': 9, '10': 'message'},

],

};

/// Descriptor for `NlpRequest`. Decode as a `google.protobuf.DescriptorProto`.

final $typed\_data.Uint8List nlpRequestDescriptor = $convert.base64Decode(

'CgpObHBSZXF1ZXN0EhgKB21lc3NhZ2UYASABKAlSB21lc3NhZ2U=');

@$core.Deprecated('Use nlpReplyDescriptor instead')

const NlpReply$json = {

'1': 'NlpReply',

'2': [

{'1': 'details', '3': 1, '4': 3, '5': 11, '6': '.nlp.Detail', '10': 'details'},

],

};

/// Descriptor for `NlpReply`. Decode as a `google.protobuf.DescriptorProto`.

final $typed\_data.Uint8List nlpReplyDescriptor = $convert.base64Decode(

'CghObHBSZXBseRIlCgdkZXRhaWxzGAEgAygLMgsubmxwLkRldGFpbFIHZGV0YWlscw==');

@$core.Deprecated('Use detailDescriptor instead')

const Detail$json = {

'1': 'Detail',

'2': [

{'1': 'name', '3': 1, '4': 1, '5': 9, '10': 'name'},

{'1': 'type', '3': 2, '4': 1, '5': 9, '10': 'type'},

],

};

/// Descriptor for `Detail`. Decode as a `google.protobuf.DescriptorProto`.

final $typed\_data.Uint8List detailDescriptor = $convert.base64Decode(

'CgZEZXRhaWwSEgoEbmFtZRgBIAEoCVIEbmFtZRISCgR0eXBlGAIgASgJUgR0eXBl');

// AUTO GENERATED FILE, DO NOT EDIT.

// Generated by `flutter\_rust\_bridge`@ 1.82.3.

// ignore\_for\_file: non\_constant\_identifier\_names, unused\_element, duplicate\_ignore, directives\_ordering, curly\_braces\_in\_flow\_control\_structures, unnecessary\_lambdas, slash\_for\_doc\_comments, prefer\_const\_literals\_to\_create\_immutables, implicit\_dynamic\_list\_literal, duplicate\_import, unused\_import, unnecessary\_import, prefer\_single\_quotes, prefer\_const\_constructors, use\_super\_parameters, always\_use\_package\_imports, annotate\_overrides, invalid\_use\_of\_protected\_member, constant\_identifier\_names, invalid\_use\_of\_internal\_member, prefer\_is\_empty, unnecessary\_const

import 'dart:convert';

import 'dart:async';

import 'package:meta/meta.dart';

import 'package:flutter\_rust\_bridge/flutter\_rust\_bridge.dart';

import 'package:uuid/uuid.dart';

import 'dart:ffi' as ffi;

abstract class Native {

Future<String> hello({dynamic hint});

FlutterRustBridgeTaskConstMeta get kHelloConstMeta;

Stream<List<FilescanResult>> localFileDetailsStream({dynamic hint});

FlutterRustBridgeTaskConstMeta get kLocalFileDetailsStreamConstMeta;

Stream<int> fileCountStream({dynamic hint});

FlutterRustBridgeTaskConstMeta get kFileCountStreamConstMeta;

Stream<bool> scanStatusStream({dynamic hint});

FlutterRustBridgeTaskConstMeta get kScanStatusStreamConstMeta;

Future<void> startScan({required String p, dynamic hint});

FlutterRustBridgeTaskConstMeta get kStartScanConstMeta;

Future<void> startLocalScan({required String p, dynamic hint});

FlutterRustBridgeTaskConstMeta get kStartLocalScanConstMeta;

Future<List<FileCount>> getScanSummary({String? path, dynamic hint});

FlutterRustBridgeTaskConstMeta get kGetScanSummaryConstMeta;

}

class FileCount {

final String fileType;

final int count;

const FileCount({

required this.fileType,

required this.count,

});

}

class FilescanResult {

final String filePath;

final String fileType;

final int updateTime;

final int createTime;

final int filesize;

final String folder;

const FilescanResult({

required this.filePath,

required this.fileType,

required this.updateTime,

required this.createTime,

required this.filesize,

required this.folder,

});

}

class NativeImpl implements Native {

final NativePlatform \_platform;

factory NativeImpl(ExternalLibrary dylib) =>

NativeImpl.raw(NativePlatform(dylib));

/// Only valid on web/WASM platforms.

factory NativeImpl.wasm(FutureOr<WasmModule> module) =>

NativeImpl(module as ExternalLibrary);

NativeImpl.raw(this.\_platform);

Future<String> hello({dynamic hint}) {

return \_platform.executeNormal(FlutterRustBridgeTask(

callFfi: (port\_) => \_platform.inner.wire\_hello(port\_),

parseSuccessData: \_wire2api\_String,

parseErrorData: null,

constMeta: kHelloConstMeta,

argValues: [],

hint: hint,

));

}

FlutterRustBridgeTaskConstMeta get kHelloConstMeta =>

const FlutterRustBridgeTaskConstMeta(

debugName: "hello",

argNames: [],

);

Stream<List<FilescanResult>> localFileDetailsStream({dynamic hint}) {

return \_platform.executeStream(FlutterRustBridgeTask(

callFfi: (port\_) => \_platform.inner.wire\_local\_file\_details\_stream(port\_),

parseSuccessData: \_wire2api\_list\_filescan\_result,

parseErrorData: \_wire2api\_FrbAnyhowException,

constMeta: kLocalFileDetailsStreamConstMeta,

argValues: [],

hint: hint,

));

}

FlutterRustBridgeTaskConstMeta get kLocalFileDetailsStreamConstMeta =>

const FlutterRustBridgeTaskConstMeta(

debugName: "local\_file\_details\_stream",

argNames: [],

);

Stream<int> fileCountStream({dynamic hint}) {

return \_platform.executeStream(FlutterRustBridgeTask(

callFfi: (port\_) => \_platform.inner.wire\_file\_count\_stream(port\_),

parseSuccessData: \_wire2api\_u64,

parseErrorData: \_wire2api\_FrbAnyhowException,

constMeta: kFileCountStreamConstMeta,

argValues: [],

hint: hint,

));

}

FlutterRustBridgeTaskConstMeta get kFileCountStreamConstMeta =>

const FlutterRustBridgeTaskConstMeta(

debugName: "file\_count\_stream",

argNames: [],

);

Stream<bool> scanStatusStream({dynamic hint}) {

return \_platform.executeStream(FlutterRustBridgeTask(

callFfi: (port\_) => \_platform.inner.wire\_scan\_status\_stream(port\_),

parseSuccessData: \_wire2api\_bool,

parseErrorData: \_wire2api\_FrbAnyhowException,

constMeta: kScanStatusStreamConstMeta,

argValues: [],

hint: hint,

));

}

FlutterRustBridgeTaskConstMeta get kScanStatusStreamConstMeta =>

const FlutterRustBridgeTaskConstMeta(

debugName: "scan\_status\_stream",

argNames: [],

);

Future<void> startScan({required String p, dynamic hint}) {

var arg0 = \_platform.api2wire\_String(p);

return \_platform.executeNormal(FlutterRustBridgeTask(

callFfi: (port\_) => \_platform.inner.wire\_start\_scan(port\_, arg0),

parseSuccessData: \_wire2api\_unit,

parseErrorData: null,

constMeta: kStartScanConstMeta,

argValues: [p],

hint: hint,

));

}

FlutterRustBridgeTaskConstMeta get kStartScanConstMeta =>

const FlutterRustBridgeTaskConstMeta(

debugName: "start\_scan",

argNames: ["p"],

);

Future<void> startLocalScan({required String p, dynamic hint}) {

var arg0 = \_platform.api2wire\_String(p);

return \_platform.executeNormal(FlutterRustBridgeTask(

callFfi: (port\_) => \_platform.inner.wire\_start\_local\_scan(port\_, arg0),

parseSuccessData: \_wire2api\_unit,

parseErrorData: null,

constMeta: kStartLocalScanConstMeta,

argValues: [p],

hint: hint,

));

}

FlutterRustBridgeTaskConstMeta get kStartLocalScanConstMeta =>

const FlutterRustBridgeTaskConstMeta(

debugName: "start\_local\_scan",

argNames: ["p"],

);

Future<List<FileCount>> getScanSummary({String? path, dynamic hint}) {

var arg0 = \_platform.api2wire\_opt\_String(path);

return \_platform.executeNormal(FlutterRustBridgeTask(

callFfi: (port\_) => \_platform.inner.wire\_get\_scan\_summary(port\_, arg0),

parseSuccessData: \_wire2api\_list\_file\_count,

parseErrorData: null,

constMeta: kGetScanSummaryConstMeta,

argValues: [path],

hint: hint,

));

}

FlutterRustBridgeTaskConstMeta get kGetScanSummaryConstMeta =>

const FlutterRustBridgeTaskConstMeta(

debugName: "get\_scan\_summary",

argNames: ["path"],

);

void dispose() {

\_platform.dispose();

}

// Section: wire2api

FrbAnyhowException \_wire2api\_FrbAnyhowException(dynamic raw) {

return FrbAnyhowException(raw as String);

}

String \_wire2api\_String(dynamic raw) {

return raw as String;

}

bool \_wire2api\_bool(dynamic raw) {

return raw as bool;

}

FileCount \_wire2api\_file\_count(dynamic raw) {

final arr = raw as List<dynamic>;

if (arr.length != 2)

throw Exception('unexpected arr length: expect 2 but see ${arr.length}');

return FileCount(

fileType: \_wire2api\_String(arr[0]),

count: \_wire2api\_u64(arr[1]),

);

}

FilescanResult \_wire2api\_filescan\_result(dynamic raw) {

final arr = raw as List<dynamic>;

if (arr.length != 6)

throw Exception('unexpected arr length: expect 6 but see ${arr.length}');

return FilescanResult(

filePath: \_wire2api\_String(arr[0]),

fileType: \_wire2api\_String(arr[1]),

updateTime: \_wire2api\_u64(arr[2]),

createTime: \_wire2api\_u64(arr[3]),

filesize: \_wire2api\_u64(arr[4]),

folder: \_wire2api\_String(arr[5]),

);

}

List<FileCount> \_wire2api\_list\_file\_count(dynamic raw) {

return (raw as List<dynamic>).map(\_wire2api\_file\_count).toList();

}

List<FilescanResult> \_wire2api\_list\_filescan\_result(dynamic raw) {

return (raw as List<dynamic>).map(\_wire2api\_filescan\_result).toList();

}

int \_wire2api\_u64(dynamic raw) {

return castInt(raw);

}

int \_wire2api\_u8(dynamic raw) {

return raw as int;

}

Uint8List \_wire2api\_uint\_8\_list(dynamic raw) {

return raw as Uint8List;

}

void \_wire2api\_unit(dynamic raw) {

return;

}

}

// Section: api2wire

@protected

int api2wire\_u8(int raw) {

return raw;

}

// Section: finalizer

class NativePlatform extends FlutterRustBridgeBase<NativeWire> {

NativePlatform(ffi.DynamicLibrary dylib) : super(NativeWire(dylib));

// Section: api2wire

@protected

ffi.Pointer<wire\_uint\_8\_list> api2wire\_String(String raw) {

return api2wire\_uint\_8\_list(utf8.encoder.convert(raw));

}

@protected

ffi.Pointer<wire\_uint\_8\_list> api2wire\_opt\_String(String? raw) {

return raw == null ? ffi.nullptr : api2wire\_String(raw);

}

@protected

ffi.Pointer<wire\_uint\_8\_list> api2wire\_uint\_8\_list(Uint8List raw) {

final ans = inner.new\_uint\_8\_list\_0(raw.length);

ans.ref.ptr.asTypedList(raw.length).setAll(0, raw);

return ans;

}

// Section: finalizer

// Section: api\_fill\_to\_wire

}

// ignore\_for\_file: camel\_case\_types, non\_constant\_identifier\_names, avoid\_positional\_boolean\_parameters, annotate\_overrides, constant\_identifier\_names

// AUTO GENERATED FILE, DO NOT EDIT.

//

// Generated by `package:ffigen`.

// ignore\_for\_file: type=lint

/// generated by flutter\_rust\_bridge

class NativeWire implements FlutterRustBridgeWireBase {

@internal

late final dartApi = DartApiDl(init\_frb\_dart\_api\_dl);

/// Holds the symbol lookup function.

final ffi.Pointer<T> Function<T extends ffi.NativeType>(String symbolName)

\_lookup;

/// The symbols are looked up in [dynamicLibrary].

NativeWire(ffi.DynamicLibrary dynamicLibrary)

: \_lookup = dynamicLibrary.lookup;

/// The symbols are looked up with [lookup].

NativeWire.fromLookup(

ffi.Pointer<T> Function<T extends ffi.NativeType>(String symbolName)

lookup)

: \_lookup = lookup;

void store\_dart\_post\_cobject(

DartPostCObjectFnType ptr,

) {

return \_store\_dart\_post\_cobject(

ptr,

);

}

late final \_store\_dart\_post\_cobjectPtr =

\_lookup<ffi.NativeFunction<ffi.Void Function(DartPostCObjectFnType)>>(

'store\_dart\_post\_cobject');

late final \_store\_dart\_post\_cobject = \_store\_dart\_post\_cobjectPtr

.asFunction<void Function(DartPostCObjectFnType)>();

Object get\_dart\_object(

int ptr,

) {

return \_get\_dart\_object(

ptr,

);

}

late final \_get\_dart\_objectPtr =

\_lookup<ffi.NativeFunction<ffi.Handle Function(ffi.UintPtr)>>(

'get\_dart\_object');

late final \_get\_dart\_object =

\_get\_dart\_objectPtr.asFunction<Object Function(int)>();

void drop\_dart\_object(

int ptr,

) {

return \_drop\_dart\_object(

ptr,

);

}

late final \_drop\_dart\_objectPtr =

\_lookup<ffi.NativeFunction<ffi.Void Function(ffi.UintPtr)>>(

'drop\_dart\_object');

late final \_drop\_dart\_object =

\_drop\_dart\_objectPtr.asFunction<void Function(int)>();

int new\_dart\_opaque(

Object handle,

) {

return \_new\_dart\_opaque(

handle,

);

}

late final \_new\_dart\_opaquePtr =

\_lookup<ffi.NativeFunction<ffi.UintPtr Function(ffi.Handle)>>(

'new\_dart\_opaque');

late final \_new\_dart\_opaque =

\_new\_dart\_opaquePtr.asFunction<int Function(Object)>();

int init\_frb\_dart\_api\_dl(

ffi.Pointer<ffi.Void> obj,

) {

return \_init\_frb\_dart\_api\_dl(

obj,

);

}

late final \_init\_frb\_dart\_api\_dlPtr =

\_lookup<ffi.NativeFunction<ffi.IntPtr Function(ffi.Pointer<ffi.Void>)>>(

'init\_frb\_dart\_api\_dl');

late final \_init\_frb\_dart\_api\_dl = \_init\_frb\_dart\_api\_dlPtr

.asFunction<int Function(ffi.Pointer<ffi.Void>)>();

void wire\_hello(

int port\_,

) {

return \_wire\_hello(

port\_,

);

}

late final \_wire\_helloPtr =

\_lookup<ffi.NativeFunction<ffi.Void Function(ffi.Int64)>>('wire\_hello');

late final \_wire\_hello = \_wire\_helloPtr.asFunction<void Function(int)>();

void wire\_local\_file\_details\_stream(

int port\_,

) {

return \_wire\_local\_file\_details\_stream(

port\_,

);

}

late final \_wire\_local\_file\_details\_streamPtr =

\_lookup<ffi.NativeFunction<ffi.Void Function(ffi.Int64)>>(

'wire\_local\_file\_details\_stream');

late final \_wire\_local\_file\_details\_stream =

\_wire\_local\_file\_details\_streamPtr.asFunction<void Function(int)>();

void wire\_file\_count\_stream(

int port\_,

) {

return \_wire\_file\_count\_stream(

port\_,

);

}

late final \_wire\_file\_count\_streamPtr =

\_lookup<ffi.NativeFunction<ffi.Void Function(ffi.Int64)>>(

'wire\_file\_count\_stream');

late final \_wire\_file\_count\_stream =

\_wire\_file\_count\_streamPtr.asFunction<void Function(int)>();

void wire\_scan\_status\_stream(

int port\_,

) {

return \_wire\_scan\_status\_stream(

port\_,

);

}

late final \_wire\_scan\_status\_streamPtr =

\_lookup<ffi.NativeFunction<ffi.Void Function(ffi.Int64)>>(

'wire\_scan\_status\_stream');

late final \_wire\_scan\_status\_stream =

\_wire\_scan\_status\_streamPtr.asFunction<void Function(int)>();

void wire\_start\_scan(

int port\_,

ffi.Pointer<wire\_uint\_8\_list> p,

) {

return \_wire\_start\_scan(

port\_,

p,

);

}

late final \_wire\_start\_scanPtr = \_lookup<

ffi.NativeFunction<

ffi.Void Function(

ffi.Int64, ffi.Pointer<wire\_uint\_8\_list>)>>('wire\_start\_scan');

late final \_wire\_start\_scan = \_wire\_start\_scanPtr

.asFunction<void Function(int, ffi.Pointer<wire\_uint\_8\_list>)>();

void wire\_start\_local\_scan(

int port\_,

ffi.Pointer<wire\_uint\_8\_list> p,

) {

return \_wire\_start\_local\_scan(

port\_,

p,

);

}

late final \_wire\_start\_local\_scanPtr = \_lookup<

ffi.NativeFunction<

ffi.Void Function(ffi.Int64,

ffi.Pointer<wire\_uint\_8\_list>)>>('wire\_start\_local\_scan');

late final \_wire\_start\_local\_scan = \_wire\_start\_local\_scanPtr

.asFunction<void Function(int, ffi.Pointer<wire\_uint\_8\_list>)>();

void wire\_get\_scan\_summary(

int port\_,

ffi.Pointer<wire\_uint\_8\_list> path,

) {

return \_wire\_get\_scan\_summary(

port\_,

path,

);

}

late final \_wire\_get\_scan\_summaryPtr = \_lookup<

ffi.NativeFunction<

ffi.Void Function(ffi.Int64,

ffi.Pointer<wire\_uint\_8\_list>)>>('wire\_get\_scan\_summary');

late final \_wire\_get\_scan\_summary = \_wire\_get\_scan\_summaryPtr

.asFunction<void Function(int, ffi.Pointer<wire\_uint\_8\_list>)>();

ffi.Pointer<wire\_uint\_8\_list> new\_uint\_8\_list\_0(

int len,

) {

return \_new\_uint\_8\_list\_0(

len,

);

}

late final \_new\_uint\_8\_list\_0Ptr = \_lookup<

ffi

.NativeFunction<ffi.Pointer<wire\_uint\_8\_list> Function(ffi.Int32)>>(

'new\_uint\_8\_list\_0');

late final \_new\_uint\_8\_list\_0 = \_new\_uint\_8\_list\_0Ptr

.asFunction<ffi.Pointer<wire\_uint\_8\_list> Function(int)>();

void free\_WireSyncReturn(

WireSyncReturn ptr,

) {

return \_free\_WireSyncReturn(

ptr,

);

}

late final \_free\_WireSyncReturnPtr =

\_lookup<ffi.NativeFunction<ffi.Void Function(WireSyncReturn)>>(

'free\_WireSyncReturn');

late final \_free\_WireSyncReturn =

\_free\_WireSyncReturnPtr.asFunction<void Function(WireSyncReturn)>();

}

final class \_Dart\_Handle extends ffi.Opaque {}

final class wire\_uint\_8\_list extends ffi.Struct {

external ffi.Pointer<ffi.Uint8> ptr;

@ffi.Int32()

external int len;

}

typedef DartPostCObjectFnType = ffi.Pointer<

ffi.NativeFunction<

ffi.Bool Function(DartPort port\_id, ffi.Pointer<ffi.Void> message)>>;

typedef DartPort = ffi.Int64;

import 'dart:ffi';

import 'bridge\_generated.dart';

// Re-export the bridge so it is only necessary to import this file.

export 'bridge\_generated.dart';

import 'dart:io' as io;

const \_base = 'native';

// On MacOS, the dynamic library is not bundled with the binary,

// but rather directly \*\*linked\*\* against the binary.

final \_dylib = io.Platform.isWindows ? '$\_base.dll' : 'lib$\_base.so';

final api = NativeImpl(io.Platform.isIOS || io.Platform.isMacOS

? DynamicLibrary.executable()

: DynamicLibrary.open(\_dylib));

import 'package:flutter/material.dart';

class HighlightText {

HighlightText.\_();

///高亮某些文字

static const TextStyle lightTextStyle = TextStyle(

color: Colors.blue,

fontWeight: FontWeight.bold,

);

/// 后续多个关键字高亮，参考 test\split\_test.dart

static InlineSpan formSpan(String src, String pattern) {

src = src.replaceAll("\n", "...");

List<TextSpan> span = [];

List<String> parts = src.split(pattern);

if (parts.length > 1) {

for (int i = 0; i < parts.length; i++) {

if (parts[i].length <= 20) {

span.add(TextSpan(text: parts[i]));

} else {

String left = parts[i].substring(0, 10);

String right = parts[i].substring(parts[i].length - 10);

String r = "$left...$right";

span.add(TextSpan(text: r));

}

if (i != parts.length - 1) {

span.add(TextSpan(text: pattern, style: lightTextStyle));

}

}

} else {

span.add(TextSpan(text: src));

}

return TextSpan(children: span);

}

static InlineSpan multiKeywords(String src, List<String> keywords) {

if (keywords.isEmpty) {

return TextSpan(text: src);

}

Set<(int, int)> positions = {};

for (final i in keywords) {

if (i == "") {

continue;

}

try {

RegExp reg = RegExp(i);

Iterable<RegExpMatch> matches = reg.allMatches(src);

for (var element in matches) {

positions.add((element.start, element.end));

}

} catch (\_) {

continue;

}

// final l = src.indexOf(i);

}

if (positions.isEmpty) {

return TextSpan(text: src);

}

List<(int, int)> p0 = positions.toList()

..sort((a, b) => a.$1.compareTo(b.$1));

// print(p0);

final set0 = \_mergeList(p0);

// print(set0);

final List<TextSpan> spans = <TextSpan>[];

int currentPosition = 0;

for (final i in set0) {

if (currentPosition != i.$1) {

spans.add(TextSpan(text: src.substring(currentPosition, i.$1)));

}

spans.add(TextSpan(

style: lightTextStyle,

text: src.substring(i.$1, i.$2),

// recognizer: span.recognizer,

));

currentPosition = i.$2;

}

if (currentPosition != src.length) {

spans.add(TextSpan(

text: src.substring(

currentPosition,

)));

}

// print(spans.length);

return TextSpan(children: spans);

}

static Set<(int, int)> \_mergeList(List<(int, int)> list) {

Set<(int, int)> results = {};

for (int i = 0; i < list.length - 1; i++) {

if (list[i].$1 == list[i + 1].$1) {

continue;

}

if (list[i].$2 <= list[i + 1].$1) {

results.add(list[i]);

} else {

list[i] = (

list[i].$1,

list[i].$2 > list[i + 1].$2 ? list[i].$2 : list[i + 1].$2

);

list[i + 1] = list[i];

results.add(list[i]);

}

}

results.add(list.last);

return results;

}

}

// ignore\_for\_file: avoid\_print, constant\_identifier\_names

import 'package:grpc/grpc.dart';

import 'package:minum\_scanner/protos/diff.pbgrpc.dart';

import 'package:minum\_scanner/protos/nlp.pbgrpc.dart';

const NEED\_NLP\_TYPES = ["pdf", "doc", "docx", "xls", "xlsx", "txt"];

const TYPE\_MAP = {

"nr": "人名",

"ns": "地名",

"nt": "组织机构名",

"nz": "专有名词",

"email": "email",

"id": "身份证号",

"ip": "ip",

"tel": "电话号码",

"url": "url",

"x": "特殊字符串"

};

class NlpTool {

NlpTool.\_();

static Future<(String, double)> getDiffResults(

String message1, String message2) async {

final channel = ClientChannel(

'localhost',

port: 15555,

options: ChannelOptions(

credentials: const ChannelCredentials.insecure(),

codecRegistry:

CodecRegistry(codecs: const [GzipCodec(), IdentityCodec()]),

),

);

final stub = DiffClient(channel);

var results = ("", 0.0);

try {

final response = await stub.getResult(DiffRequest()

..text1 = message1

..text2 = message2);

results = (response.message, response.similarity);

} catch (e) {

print(e);

}

await channel.shutdown();

return results;

}

static Future<List<Detail>> getJiojioResults(String message) async {

final channel = ClientChannel(

'localhost',

port: 15555,

options: ChannelOptions(

credentials: const ChannelCredentials.insecure(),

codecRegistry:

CodecRegistry(codecs: const [GzipCodec(), IdentityCodec()]),

),

);

final stub = NlpExtractorClient(channel);

List<Detail> results = [];

try {

final response = await stub.getResult(NlpRequest()..message = message);

results = response.details;

} catch (e) {

print(e);

}

await channel.shutdown();

return results;

}

}

extension DetailGrep on List<Detail> {

Map<String, int> grep() {

final Map<String, int> result = {};

for (final i in this) {

if (TYPE\_MAP[i.type] == null) {

continue;

}

if (result[TYPE\_MAP[i.type]] != null) {

result[TYPE\_MAP[i.type]!] = result[TYPE\_MAP[i.type]]! + 1;

} else {

result[TYPE\_MAP[i.type]!] = 1;

}

}

return result;

}

Map<String, List<String>> grepWithValues() {

final Map<String, Set<String>> result = {};

for (final i in this) {

if (TYPE\_MAP[i.type] == null) {

continue;

}

if (i.name == "\n") {

continue;

}

if (result[TYPE\_MAP[i.type]] != null) {

result[TYPE\_MAP[i.type]!]!.add(i.name);

} else {

result[TYPE\_MAP[i.type]!] = {i.name};

}

}

return {}

..addAll(result.map((key, value) => MapEntry(key, value.toList())));

}

}

import 'package:dio/dio.dart';

import 'package:flutter/foundation.dart';

class PostPathRequest {

String? path;

String? ext;

PostPathRequest({this.path, this.ext});

PostPathRequest.fromJson(Map<String, dynamic> json) {

path = json['path'];

ext = json['ext'];

}

Map<String, dynamic> toJson() {

final Map<String, dynamic> data = <String, dynamic>{};

data['path'] = path;

data['ext'] = ext;

return data;

}

}

class PostPathResponse {

String? content;

PostPathResponse({this.content});

PostPathResponse.fromJson(Map<String, dynamic> json) {

content = json['content'];

}

Map<String, dynamic> toJson() {

final Map<String, dynamic> data = <String, dynamic>{};

data['content'] = content;

return data;

}

}

class TikaTool {

TikaTool.\_();

static final Dio \_dio = Dio();

static const String \_url = "http://localhost:15433/file/detect";

static Future<String> postToTika(String f, String ext) async {

try {

Response response = await \_dio.post(\_url,

data: PostPathRequest(ext: ext, path: f).toJson());

final r = PostPathResponse.fromJson(response.data['data']);

return r.content ?? "";

} catch (e) {

if (kDebugMode) {

print(e);

}

}

return "";

}

}

import 'package:flutter/material.dart';

import 'package:flutter\_riverpod/flutter\_riverpod.dart';

import 'package:isar/isar.dart';

import 'package:minum\_scanner/common/nlp.dart';

import 'package:minum\_scanner/isar/database.dart';

import 'package:minum\_scanner/isar/file\_details.dart';

import 'package:pretty\_diff\_text/pretty\_diff\_text.dart';

class CompareScreen extends ConsumerStatefulWidget {

const CompareScreen({super.key});

@override

ConsumerState<CompareScreen> createState() => \_CompareScreenState();

}

class \_CompareScreenState extends ConsumerState<CompareScreen> {

// ignore: prefer\_typing\_uninitialized\_variables

late Future<(String, double)> future;

final IsarDatabase isarDatabase = IsarDatabase();

late List<FileDetails> l = [];

Future<(String, double)> init() async {

/// for test

l = await isarDatabase.isar!.fileDetails.where().limit(2).findAll();

final results = await NlpTool.getDiffResults(l[0].content!, l[1].content!);

return results;

}

@override

void initState() {

super.initState();

future = init();

}

@override

Widget build(BuildContext context) {

return Container(

decoration: BoxDecoration(

color: Colors.white,

borderRadius: BorderRadius.circular(15),

boxShadow: [

BoxShadow(

color: Colors.black.withAlpha(25),

offset: const Offset(4, 4),

blurRadius: 5,

spreadRadius: 5,

),

],

),

width: 0.8 \* (MediaQuery.of(context).size.width - 200),

height: 0.8 \* MediaQuery.of(context).size.height,

child: FutureBuilder<(String, double)>(

future: future,

builder: (c, s) {

if (s.connectionState == ConnectionState.done) {

return Column(

children: [

SizedBox(

height: 30,

child: Text("相似度: ${s.data!.$2}"),

),

Expanded(

child: SingleChildScrollView(

child: PrettyDiffText(

oldText: l[0].content ?? "",

newText: l[1].content ?? "",

),

))

],

);

}

return const Center(

child: CircularProgressIndicator(),

);

}),

);

}

}

// GENERATED CODE - DO NOT MODIFY BY HAND

part of 'file\_details.dart';

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// IsarCollectionGenerator

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// coverage:ignore-file

// ignore\_for\_file: duplicate\_ignore, non\_constant\_identifier\_names, constant\_identifier\_names, invalid\_use\_of\_protected\_member, unnecessary\_cast, prefer\_const\_constructors, lines\_longer\_than\_80\_chars, require\_trailing\_commas, inference\_failure\_on\_function\_invocation, unnecessary\_parenthesis, unnecessary\_raw\_strings, unnecessary\_null\_checks, join\_return\_with\_assignment, prefer\_final\_locals, avoid\_js\_rounded\_ints, avoid\_positional\_boolean\_parameters, always\_specify\_types

extension GetFileDetailsCollection on Isar {

IsarCollection<FileDetails> get fileDetails => this.collection();

}

const FileDetailsSchema = CollectionSchema(

name: r'FileDetails',

id: -6838175689154942898,

properties: {

r'content': PropertySchema(

id: 0,

name: r'content',

type: IsarType.string,

),

r'createTime': PropertySchema(

id: 1,

name: r'createTime',

type: IsarType.long,

),

r'detected': PropertySchema(

id: 2,

name: r'detected',

type: IsarType.bool,

),

r'filePath': PropertySchema(

id: 3,

name: r'filePath',

type: IsarType.string,

),

r'fileType': PropertySchema(

id: 4,

name: r'fileType',

type: IsarType.string,

),

r'filesize': PropertySchema(

id: 5,

name: r'filesize',

type: IsarType.long,

),

r'keywords': PropertySchema(

id: 6,

name: r'keywords',

type: IsarType.string,

),

r'rating': PropertySchema(

id: 7,

name: r'rating',

type: IsarType.double,

),

r'updateTime': PropertySchema(

id: 8,

name: r'updateTime',

type: IsarType.long,

)

},

estimateSize: \_fileDetailsEstimateSize,

serialize: \_fileDetailsSerialize,

deserialize: \_fileDetailsDeserialize,

deserializeProp: \_fileDetailsDeserializeProp,

idName: r'detailId',

indexes: {},

links: {},

embeddedSchemas: {},

getId: \_fileDetailsGetId,

getLinks: \_fileDetailsGetLinks,

attach: \_fileDetailsAttach,

version: '3.1.0+1',

);

int \_fileDetailsEstimateSize(

FileDetails object,

List<int> offsets,

Map<Type, List<int>> allOffsets,

) {

var bytesCount = offsets.last;

{

final value = object.content;

if (value != null) {

bytesCount += 3 + value.length \* 3;

}

}

{

final value = object.filePath;

if (value != null) {

bytesCount += 3 + value.length \* 3;

}

}

{

final value = object.fileType;

if (value != null) {

bytesCount += 3 + value.length \* 3;

}

}

{

final value = object.keywords;

if (value != null) {

bytesCount += 3 + value.length \* 3;

}

}

return bytesCount;

}

void \_fileDetailsSerialize(

FileDetails object,

IsarWriter writer,

List<int> offsets,

Map<Type, List<int>> allOffsets,

) {

writer.writeString(offsets[0], object.content);

writer.writeLong(offsets[1], object.createTime);

writer.writeBool(offsets[2], object.detected);

writer.writeString(offsets[3], object.filePath);

writer.writeString(offsets[4], object.fileType);

writer.writeLong(offsets[5], object.filesize);

writer.writeString(offsets[6], object.keywords);

writer.writeDouble(offsets[7], object.rating);

writer.writeLong(offsets[8], object.updateTime);

}

FileDetails \_fileDetailsDeserialize(

Id id,

IsarReader reader,

List<int> offsets,

Map<Type, List<int>> allOffsets,

) {

final object = FileDetails();

object.content = reader.readStringOrNull(offsets[0]);

object.createTime = reader.readLongOrNull(offsets[1]);

object.detailId = id;

object.detected = reader.readBool(offsets[2]);

object.filePath = reader.readStringOrNull(offsets[3]);

object.fileType = reader.readStringOrNull(offsets[4]);

object.filesize = reader.readLongOrNull(offsets[5]);

object.keywords = reader.readStringOrNull(offsets[6]);

object.rating = reader.readDoubleOrNull(offsets[7]);

object.updateTime = reader.readLongOrNull(offsets[8]);

return object;

}

P \_fileDetailsDeserializeProp<P>(

IsarReader reader,

int propertyId,

int offset,

Map<Type, List<int>> allOffsets,

) {

switch (propertyId) {

case 0:

return (reader.readStringOrNull(offset)) as P;

case 1:

return (reader.readLongOrNull(offset)) as P;

case 2:

return (reader.readBool(offset)) as P;

case 3:

return (reader.readStringOrNull(offset)) as P;

case 4:

return (reader.readStringOrNull(offset)) as P;

case 5:

return (reader.readLongOrNull(offset)) as P;

case 6:

return (reader.readStringOrNull(offset)) as P;

case 7:

return (reader.readDoubleOrNull(offset)) as P;

case 8:

return (reader.readLongOrNull(offset)) as P;

default:

throw IsarError('Unknown property with id $propertyId');

}

}

Id \_fileDetailsGetId(FileDetails object) {

return object.detailId;

}

List<IsarLinkBase<dynamic>> \_fileDetailsGetLinks(FileDetails object) {

return [];

}

void \_fileDetailsAttach(

IsarCollection<dynamic> col, Id id, FileDetails object) {

object.detailId = id;

}

extension FileDetailsQueryWhereSort

on QueryBuilder<FileDetails, FileDetails, QWhere> {

QueryBuilder<FileDetails, FileDetails, QAfterWhere> anyDetailId() {

return QueryBuilder.apply(this, (query) {

return query.addWhereClause(const IdWhereClause.any());

});

}

}

extension FileDetailsQueryWhere

on QueryBuilder<FileDetails, FileDetails, QWhereClause> {

QueryBuilder<FileDetails, FileDetails, QAfterWhereClause> detailIdEqualTo(

Id detailId) {

return QueryBuilder.apply(this, (query) {

return query.addWhereClause(IdWhereClause.between(

lower: detailId,

upper: detailId,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterWhereClause> detailIdNotEqualTo(

Id detailId) {

return QueryBuilder.apply(this, (query) {

if (query.whereSort == Sort.asc) {

return query

.addWhereClause(

IdWhereClause.lessThan(upper: detailId, includeUpper: false),

)

.addWhereClause(

IdWhereClause.greaterThan(lower: detailId, includeLower: false),

);

} else {

return query

.addWhereClause(

IdWhereClause.greaterThan(lower: detailId, includeLower: false),

)

.addWhereClause(

IdWhereClause.lessThan(upper: detailId, includeUpper: false),

);

}

});

}

QueryBuilder<FileDetails, FileDetails, QAfterWhereClause> detailIdGreaterThan(

Id detailId,

{bool include = false}) {

return QueryBuilder.apply(this, (query) {

return query.addWhereClause(

IdWhereClause.greaterThan(lower: detailId, includeLower: include),

);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterWhereClause> detailIdLessThan(

Id detailId,

{bool include = false}) {

return QueryBuilder.apply(this, (query) {

return query.addWhereClause(

IdWhereClause.lessThan(upper: detailId, includeUpper: include),

);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterWhereClause> detailIdBetween(

Id lowerDetailId,

Id upperDetailId, {

bool includeLower = true,

bool includeUpper = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addWhereClause(IdWhereClause.between(

lower: lowerDetailId,

includeLower: includeLower,

upper: upperDetailId,

includeUpper: includeUpper,

));

});

}

}

extension FileDetailsQueryFilter

on QueryBuilder<FileDetails, FileDetails, QFilterCondition> {

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

contentIsNull() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(const FilterCondition.isNull(

property: r'content',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

contentIsNotNull() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(const FilterCondition.isNotNull(

property: r'content',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> contentEqualTo(

String? value, {

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.equalTo(

property: r'content',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

contentGreaterThan(

String? value, {

bool include = false,

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.greaterThan(

include: include,

property: r'content',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> contentLessThan(

String? value, {

bool include = false,

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.lessThan(

include: include,

property: r'content',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> contentBetween(

String? lower,

String? upper, {

bool includeLower = true,

bool includeUpper = true,

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.between(

property: r'content',

lower: lower,

includeLower: includeLower,

upper: upper,

includeUpper: includeUpper,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

contentStartsWith(

String value, {

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.startsWith(

property: r'content',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> contentEndsWith(

String value, {

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.endsWith(

property: r'content',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> contentContains(

String value,

{bool caseSensitive = true}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.contains(

property: r'content',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> contentMatches(

String pattern,

{bool caseSensitive = true}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.matches(

property: r'content',

wildcard: pattern,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

contentIsEmpty() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.equalTo(

property: r'content',

value: '',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

contentIsNotEmpty() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.greaterThan(

property: r'content',

value: '',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

createTimeIsNull() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(const FilterCondition.isNull(

property: r'createTime',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

createTimeIsNotNull() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(const FilterCondition.isNotNull(

property: r'createTime',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

createTimeEqualTo(int? value) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.equalTo(

property: r'createTime',

value: value,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

createTimeGreaterThan(

int? value, {

bool include = false,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.greaterThan(

include: include,

property: r'createTime',

value: value,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

createTimeLessThan(

int? value, {

bool include = false,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.lessThan(

include: include,

property: r'createTime',

value: value,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

createTimeBetween(

int? lower,

int? upper, {

bool includeLower = true,

bool includeUpper = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.between(

property: r'createTime',

lower: lower,

includeLower: includeLower,

upper: upper,

includeUpper: includeUpper,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> detailIdEqualTo(

Id value) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.equalTo(

property: r'detailId',

value: value,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

detailIdGreaterThan(

Id value, {

bool include = false,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.greaterThan(

include: include,

property: r'detailId',

value: value,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

detailIdLessThan(

Id value, {

bool include = false,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.lessThan(

include: include,

property: r'detailId',

value: value,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> detailIdBetween(

Id lower,

Id upper, {

bool includeLower = true,

bool includeUpper = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.between(

property: r'detailId',

lower: lower,

includeLower: includeLower,

upper: upper,

includeUpper: includeUpper,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> detectedEqualTo(

bool value) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.equalTo(

property: r'detected',

value: value,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

filePathIsNull() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(const FilterCondition.isNull(

property: r'filePath',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

filePathIsNotNull() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(const FilterCondition.isNotNull(

property: r'filePath',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> filePathEqualTo(

String? value, {

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.equalTo(

property: r'filePath',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

filePathGreaterThan(

String? value, {

bool include = false,

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.greaterThan(

include: include,

property: r'filePath',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

filePathLessThan(

String? value, {

bool include = false,

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.lessThan(

include: include,

property: r'filePath',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> filePathBetween(

String? lower,

String? upper, {

bool includeLower = true,

bool includeUpper = true,

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.between(

property: r'filePath',

lower: lower,

includeLower: includeLower,

upper: upper,

includeUpper: includeUpper,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

filePathStartsWith(

String value, {

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.startsWith(

property: r'filePath',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

filePathEndsWith(

String value, {

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.endsWith(

property: r'filePath',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

filePathContains(String value, {bool caseSensitive = true}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.contains(

property: r'filePath',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> filePathMatches(

String pattern,

{bool caseSensitive = true}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.matches(

property: r'filePath',

wildcard: pattern,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

filePathIsEmpty() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.equalTo(

property: r'filePath',

value: '',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

filePathIsNotEmpty() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.greaterThan(

property: r'filePath',

value: '',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

fileTypeIsNull() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(const FilterCondition.isNull(

property: r'fileType',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

fileTypeIsNotNull() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(const FilterCondition.isNotNull(

property: r'fileType',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> fileTypeEqualTo(

String? value, {

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.equalTo(

property: r'fileType',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

fileTypeGreaterThan(

String? value, {

bool include = false,

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.greaterThan(

include: include,

property: r'fileType',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

fileTypeLessThan(

String? value, {

bool include = false,

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.lessThan(

include: include,

property: r'fileType',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> fileTypeBetween(

String? lower,

String? upper, {

bool includeLower = true,

bool includeUpper = true,

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.between(

property: r'fileType',

lower: lower,

includeLower: includeLower,

upper: upper,

includeUpper: includeUpper,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

fileTypeStartsWith(

String value, {

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.startsWith(

property: r'fileType',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

fileTypeEndsWith(

String value, {

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.endsWith(

property: r'fileType',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

fileTypeContains(String value, {bool caseSensitive = true}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.contains(

property: r'fileType',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> fileTypeMatches(

String pattern,

{bool caseSensitive = true}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.matches(

property: r'fileType',

wildcard: pattern,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

fileTypeIsEmpty() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.equalTo(

property: r'fileType',

value: '',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

fileTypeIsNotEmpty() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.greaterThan(

property: r'fileType',

value: '',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

filesizeIsNull() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(const FilterCondition.isNull(

property: r'filesize',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

filesizeIsNotNull() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(const FilterCondition.isNotNull(

property: r'filesize',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> filesizeEqualTo(

int? value) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.equalTo(

property: r'filesize',

value: value,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

filesizeGreaterThan(

int? value, {

bool include = false,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.greaterThan(

include: include,

property: r'filesize',

value: value,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

filesizeLessThan(

int? value, {

bool include = false,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.lessThan(

include: include,

property: r'filesize',

value: value,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> filesizeBetween(

int? lower,

int? upper, {

bool includeLower = true,

bool includeUpper = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.between(

property: r'filesize',

lower: lower,

includeLower: includeLower,

upper: upper,

includeUpper: includeUpper,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

keywordsIsNull() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(const FilterCondition.isNull(

property: r'keywords',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

keywordsIsNotNull() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(const FilterCondition.isNotNull(

property: r'keywords',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> keywordsEqualTo(

String? value, {

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.equalTo(

property: r'keywords',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

keywordsGreaterThan(

String? value, {

bool include = false,

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.greaterThan(

include: include,

property: r'keywords',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

keywordsLessThan(

String? value, {

bool include = false,

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.lessThan(

include: include,

property: r'keywords',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> keywordsBetween(

String? lower,

String? upper, {

bool includeLower = true,

bool includeUpper = true,

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.between(

property: r'keywords',

lower: lower,

includeLower: includeLower,

upper: upper,

includeUpper: includeUpper,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

keywordsStartsWith(

String value, {

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.startsWith(

property: r'keywords',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

keywordsEndsWith(

String value, {

bool caseSensitive = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.endsWith(

property: r'keywords',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

keywordsContains(String value, {bool caseSensitive = true}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.contains(

property: r'keywords',

value: value,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> keywordsMatches(

String pattern,

{bool caseSensitive = true}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.matches(

property: r'keywords',

wildcard: pattern,

caseSensitive: caseSensitive,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

keywordsIsEmpty() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.equalTo(

property: r'keywords',

value: '',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

keywordsIsNotEmpty() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.greaterThan(

property: r'keywords',

value: '',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> ratingIsNull() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(const FilterCondition.isNull(

property: r'rating',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

ratingIsNotNull() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(const FilterCondition.isNotNull(

property: r'rating',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> ratingEqualTo(

double? value, {

double epsilon = Query.epsilon,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.equalTo(

property: r'rating',

value: value,

epsilon: epsilon,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

ratingGreaterThan(

double? value, {

bool include = false,

double epsilon = Query.epsilon,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.greaterThan(

include: include,

property: r'rating',

value: value,

epsilon: epsilon,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> ratingLessThan(

double? value, {

bool include = false,

double epsilon = Query.epsilon,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.lessThan(

include: include,

property: r'rating',

value: value,

epsilon: epsilon,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition> ratingBetween(

double? lower,

double? upper, {

bool includeLower = true,

bool includeUpper = true,

double epsilon = Query.epsilon,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.between(

property: r'rating',

lower: lower,

includeLower: includeLower,

upper: upper,

includeUpper: includeUpper,

epsilon: epsilon,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

updateTimeIsNull() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(const FilterCondition.isNull(

property: r'updateTime',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

updateTimeIsNotNull() {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(const FilterCondition.isNotNull(

property: r'updateTime',

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

updateTimeEqualTo(int? value) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.equalTo(

property: r'updateTime',

value: value,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

updateTimeGreaterThan(

int? value, {

bool include = false,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.greaterThan(

include: include,

property: r'updateTime',

value: value,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

updateTimeLessThan(

int? value, {

bool include = false,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.lessThan(

include: include,

property: r'updateTime',

value: value,

));

});

}

QueryBuilder<FileDetails, FileDetails, QAfterFilterCondition>

updateTimeBetween(

int? lower,

int? upper, {

bool includeLower = true,

bool includeUpper = true,

}) {

return QueryBuilder.apply(this, (query) {

return query.addFilterCondition(FilterCondition.between(

property: r'updateTime',

lower: lower,

includeLower: includeLower,

upper: upper,

includeUpper: includeUpper,

));

});

}

}

extension FileDetailsQueryObject

on QueryBuilder<FileDetails, FileDetails, QFilterCondition> {}

extension FileDetailsQueryLinks

on QueryBuilder<FileDetails, FileDetails, QFilterCondition> {}

extension FileDetailsQuerySortBy

on QueryBuilder<FileDetails, FileDetails, QSortBy> {

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> sortByContent() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'content', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> sortByContentDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'content', Sort.desc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> sortByCreateTime() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'createTime', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> sortByCreateTimeDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'createTime', Sort.desc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> sortByDetected() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'detected', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> sortByDetectedDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'detected', Sort.desc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> sortByFilePath() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'filePath', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> sortByFilePathDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'filePath', Sort.desc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> sortByFileType() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'fileType', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> sortByFileTypeDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'fileType', Sort.desc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> sortByFilesize() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'filesize', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> sortByFilesizeDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'filesize', Sort.desc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> sortByKeywords() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'keywords', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> sortByKeywordsDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'keywords', Sort.desc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> sortByRating() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'rating', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> sortByRatingDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'rating', Sort.desc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> sortByUpdateTime() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'updateTime', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> sortByUpdateTimeDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'updateTime', Sort.desc);

});

}

}

extension FileDetailsQuerySortThenBy

on QueryBuilder<FileDetails, FileDetails, QSortThenBy> {

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByContent() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'content', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByContentDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'content', Sort.desc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByCreateTime() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'createTime', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByCreateTimeDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'createTime', Sort.desc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByDetailId() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'detailId', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByDetailIdDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'detailId', Sort.desc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByDetected() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'detected', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByDetectedDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'detected', Sort.desc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByFilePath() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'filePath', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByFilePathDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'filePath', Sort.desc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByFileType() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'fileType', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByFileTypeDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'fileType', Sort.desc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByFilesize() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'filesize', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByFilesizeDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'filesize', Sort.desc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByKeywords() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'keywords', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByKeywordsDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'keywords', Sort.desc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByRating() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'rating', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByRatingDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'rating', Sort.desc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByUpdateTime() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'updateTime', Sort.asc);

});

}

QueryBuilder<FileDetails, FileDetails, QAfterSortBy> thenByUpdateTimeDesc() {

return QueryBuilder.apply(this, (query) {

return query.addSortBy(r'updateTime', Sort.desc);

});

}

}

extension FileDetailsQueryWhereDistinct

on QueryBuilder<FileDetails, FileDetails, QDistinct> {

QueryBuilder<FileDetails, FileDetails, QDistinct> distinctByContent(

{bool caseSensitive = true}) {

return QueryBuilder.apply(this, (query) {

return query.addDistinctBy(r'content', caseSensitive: caseSensitive);

});

}

QueryBuilder<FileDetails, FileDetails, QDistinct> distinctByCreateTime() {

return QueryBuilder.apply(this, (query) {

return query.addDistinctBy(r'createTime');

});

}

QueryBuilder<FileDetails, FileDetails, QDistinct> distinctByDetected() {

return QueryBuilder.apply(this, (query) {

return query.addDistinctBy(r'detected');

});

}

QueryBuilder<FileDetails, FileDetails, QDistinct> distinctByFilePath(

{bool caseSensitive = true}) {

return QueryBuilder.apply(this, (query) {

return query.addDistinctBy(r'filePath', caseSensitive: caseSensitive);

});

}

QueryBuilder<FileDetails, FileDetails, QDistinct> distinctByFileType(

{bool caseSensitive = true}) {

return QueryBuilder.apply(this, (query) {

return query.addDistinctBy(r'fileType', caseSensitive: caseSensitive);

});

}

QueryBuilder<FileDetails, FileDetails, QDistinct> distinctByFilesize() {

return QueryBuilder.apply(this, (query) {

return query.addDistinctBy(r'filesize');

});

}

QueryBuilder<FileDetails, FileDetails, QDistinct> distinctByKeywords(

{bool caseSensitive = true}) {

return QueryBuilder.apply(this, (query) {

return query.addDistinctBy(r'keywords', caseSensitive: caseSensitive);

});

}

QueryBuilder<FileDetails, FileDetails, QDistinct> distinctByRating() {

return QueryBuilder.apply(this, (query) {

return query.addDistinctBy(r'rating');

});

}

QueryBuilder<FileDetails, FileDetails, QDistinct> distinctByUpdateTime() {

return QueryBuilder.apply(this, (query) {

return query.addDistinctBy(r'updateTime');

});

}

}

extension FileDetailsQueryProperty

on QueryBuilder<FileDetails, FileDetails, QQueryProperty> {

QueryBuilder<FileDetails, int, QQueryOperations> detailIdProperty() {

return QueryBuilder.apply(this, (query) {

return query.addPropertyName(r'detailId');

});

}

QueryBuilder<FileDetails, String?, QQueryOperations> contentProperty() {

return QueryBuilder.apply(this, (query) {

return query.addPropertyName(r'content');

});

}

QueryBuilder<FileDetails, int?, QQueryOperations> createTimeProperty() {

return QueryBuilder.apply(this, (query) {

return query.addPropertyName(r'createTime');

});

}

QueryBuilder<FileDetails, bool, QQueryOperations> detectedProperty() {

return QueryBuilder.apply(this, (query) {

return query.addPropertyName(r'detected');

});

}

QueryBuilder<FileDetails, String?, QQueryOperations> filePathProperty() {

return QueryBuilder.apply(this, (query) {

return query.addPropertyName(r'filePath');

});

}

QueryBuilder<FileDetails, String?, QQueryOperations> fileTypeProperty() {

return QueryBuilder.apply(this, (query) {

return query.addPropertyName(r'fileType');

});

}

QueryBuilder<FileDetails, int?, QQueryOperations> filesizeProperty() {

return QueryBuilder.apply(this, (query) {

return query.addPropertyName(r'filesize');

});

}

QueryBuilder<FileDetails, String?, QQueryOperations> keywordsProperty() {

return QueryBuilder.apply(this, (query) {

return query.addPropertyName(r'keywords');

});

}

QueryBuilder<FileDetails, double?, QQueryOperations> ratingProperty() {

return QueryBuilder.apply(this, (query) {

return query.addPropertyName(r'rating');

});

}

QueryBuilder<FileDetails, int?, QQueryOperations> updateTimeProperty() {

return QueryBuilder.apply(this, (query) {

return query.addPropertyName(r'updateTime');

});

}

}