Sumário

Compras é um aplicativo para registrar os itens comprados em supermercados, farmacias e outros. Estes registros poderam ser comparados com compras anteriores e verificar as alterações nos preços dos produtos e de suas compras mensais.

Índice

Introdução

Compras é um projeto criado a partir de uma demanda de minha querida ex-esposa Jeani Daniela Bortolini para simplificar o controle e o acompanhamento das compras dos usuários.

Produto Mínimo Viável

Nesta primeira versão, o aplicativo irá focar em:

- Registrar os itens comprados e o valor total de cada compra.
- Monitorar variações de preço dos produtos ao longo do tempo.
- Exibir um histórico mensal de gastos e alterações de valores.

Para isso, a aplicação utilizará um banco de dados SQLite local, contendo uma lista genérica de produtos pré-cadastrados.

Versão Final

Em versões posteriores, o sistema será ampliado para:

- Registrar as compras realizadas em um servidor remoto, com autenticação de usuários e gerenciamento de contas.
- Conectar-se a uma API externa para obter dados de produtos via código de barras.
- Permitir ao usuário criar, editar e gerenciar listas de compras.
- Gerar recomendações de melhores locais de compra com base em índices de preço e perfil de consumo.
- Calcular índices de inflação dos preços dos produtos ao longo do tempo.
- Cruzar dados de diferentes consumidores em diversas regiões para análises comparativas.

Arquitetura

A aplicação seguirá o padrão **MVVM**, conforme orientação do Flutter Team, dividida em três grandes camadas:

1. Data

Responsável pelo CRUD junto à fonte de dados. Subdividida em:

- **Service** Abstrai o servidor de dados e converte o payload bruto em modelos de domínio. Essa camada isola a origem dos dados, facilitando mudanças futuras sem impactar as camadas superiores.
- **Repository** Responsável por fornecer os dados ao restante do app, orquestrando chamadas ao Service e mantendo um cache interno dos dados já carregados. O Repository também trabalhará como a *Source of True* dos dados, podendo ser reativo para mudanças nos dados armazenados.

2. Domain

Agrupa modelos e regras de negócio:

- **models** Declaração de modelos, DTOs e enums da aplicação. Utiliza o `freezed` para geração automática de Data Classes e DTOs.
- **use_cases** Classes que organizam regras de negócio mais complexas ou que envolvem múltiplos repositórios. Para casos simples, a lógica pode permanecer na ViewModel.

3. UI

Interface com o usuário e ligação com a lógica de negócio:

- **views** Componentes de interface gráfica (telas, widgets, layouts).
- **view_models** Ponte entre as Views e os Repositories/UseCases. Aqui acontece a manipulação de estado e lógica de apresentação; para fluxos muito complexos, a responsabilidade pode ser compartilhada com UseCases.

O diagrama abaixo ilustra a arquitetura proposta pelo time do Flutter MVVM.

De uma forma geral os dados serão transitados dos Repositories para as ViewModels e das ViewModels para os Repositories. Os UseCases serão usados para lidar com regras de negócio mais complexas ou que envolvem múltiplos repositórios.

A camada ViewModel irá gerenciar a reatividade por meio do emprego de um Command, encapulando as regras de negócio e a lógica de apresentação.

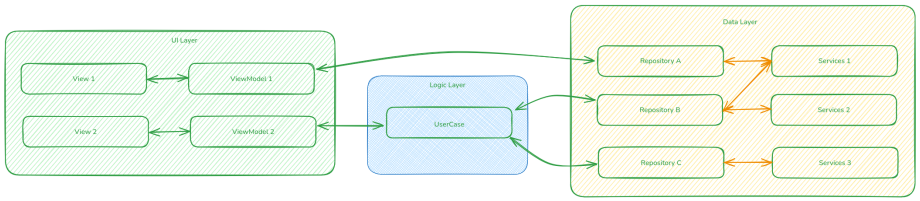

Figura 1: estrutura Básica da Arquitetura MVVM

Para o MVP (Minimum Viable Product)

Para o MVP o diagrama de classes da aplicação será implementado conforme o diagrama a seguir:

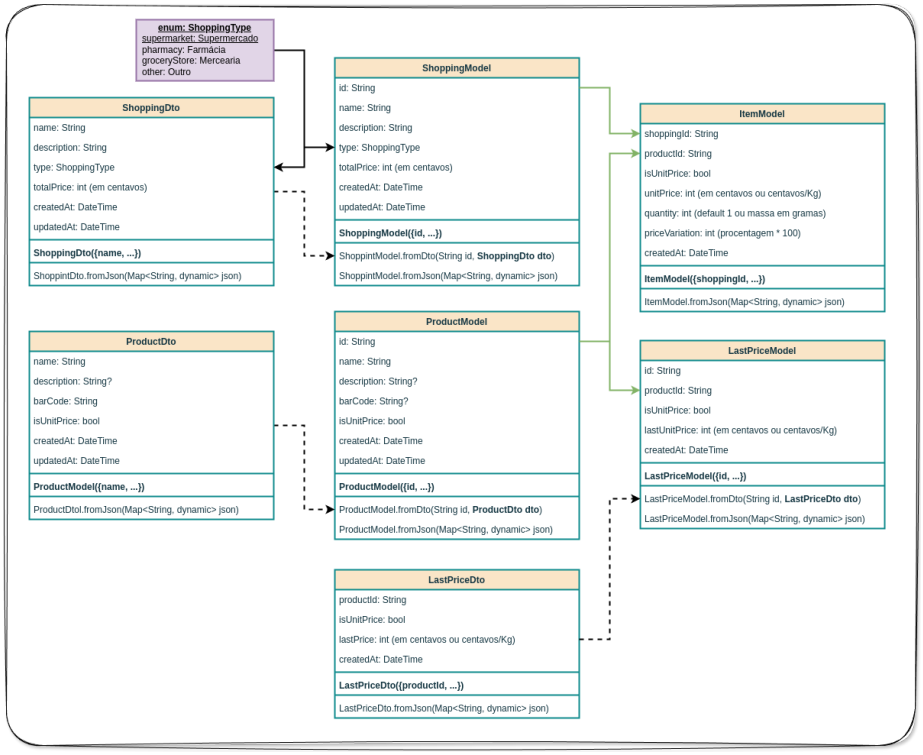

Figura 2: Diagrama de Classes

Onde as classes principais são:

1. ShoppingModel

Representa o modelo de dados da compra, contendo os dados de uma compra.

- **id** - identificador da compra;
- **name** - nome da compra;
- **description** - descrição da compra;
- **totalPrice** - valor total da compra;
- **type** - tipo da compra (mercado, farmácia, etc.);
- **createdAt** - data de criação da compra;
- **updatedAt** - data de atualização da compra;

2. ProductModel

Representa o modelo de dados do produto, contendo os dados de um produto, relacionados ao seu código de barras.

- **id** - identificador do produto;
- **name** - nome do produto;
- **description** - descrição do produto;
- **barCode** - código de barras do produto;
- **saleBy** - indica se o produto é vendido a unidade ou a quilo;
- **categoryName** - categoria do produto;
- **categoryId** - identificador da categoria do produto;
- **subCategoryName** - subcategoria do produto;
- **subCategoryId** - identificador da subcategoria do produto;
- **createdAt** - data de criação do produto;
- **updatedAt** - data de atualização do produto;

3. ItemModel

Representa o modelo de dados do item da compra, contendo os dados de um item da compra, como quantidade, preço unitário e produto.

- **shoppingId** - identificador da compra;
- **productId** - identificador do produto;
- **name** - nome do produto;
- **saleBy** - indica se o produto é vendido a unidade ou a quilo;
- **unitPrice** - preço unitário do produto;
- **quantity** - quantidade do produto;
- **priceVariation** - variação do preço do produto;
- **createdAt** - data de criação do item da compra;
- **updatedAt** - data de atualização do item da compra;

4. LastPriceModel

Esta classe será responsável por manter registros dos valores de compra dos produtos, conforme a data de compra, para facilitar a visualização de variações nos preços dos produtos ao longo do tempo.

- **id** - identificador do registro;
- **productId** - identificador do produto;

- **lastUnitPrice** - preço unitário do produto;
- **saleBy** - indica se o produto é vendido a unidade ou a quilo;
- **createdAt** - data de criação do registro;

5. CategoryModel

Representa o modelo de dados da categoria, contendo os dados de uma categoria.

- **id** - identificador da categoria;
- **name** - nome da categoria;

6. SubCategoryModel

Representa o modelo de dados da subcategoria, contendo os dados de uma subcategoria.

- **id** - identificador da subcategoria;
- **name** - nome da subcategoria;
- **categoryId** - identificador da categoria;

Category e Subcategory serão empregados para a classificação dos produtos e geração de relatórios de variação de preços.

MVVM para o MVP

Para o MVP o diagrama de classes da implementação do MVVM será implementado conforme o diagrama a seguir:

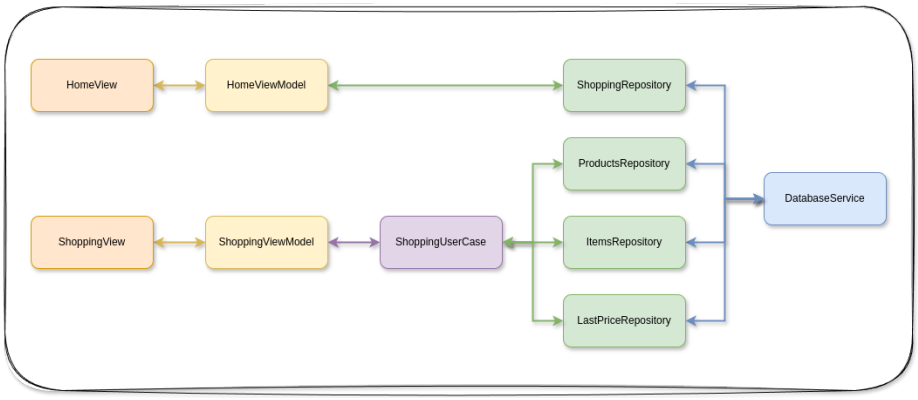

Figura 3: MVVM para o MVP

A HomeView vai manter as listas das compras registradas no aplicativo, sendo responsável por criar novas listas e administrá-las.

A ShoppingView será o responsável por:

- criar novos produtos;

- criar novos registros de preços;
- lançar itens nos carrinhos.

Por isto que este último interage com a ShoppingUserCase, para gerenciar os itens da compra nos diferentes repositórios: ProductRepository, ItemsRepository e LastPriceRepository.

Changelog

This is a list of changes made to the codebase since the last release.

2025/07/08 save_product-03 - rudsonalves

Add documentation structure and reorganize docs assets

This commit introduces the foundational documentation for the Compras project, adding key markdown files and assets. It also renames and relocates existing changelog and diagram files to maintain a consistent docs hierarchy. The changes ensure all architectural and introductory materials are available under `docs`, with images centralized in the `images` subfolder.

Modified Files

- **docs/CHANGELOG.md → docs/Changelog.md**
  - Renamed the changelog to standardize filename casing and update references.
- **docs/Diagrama_de_Classes.drawio → docs/images/Diagrama_de_Classes.drawio**
  - Moved the class diagram file into the `images` directory for better organization.
- **docs/images/MVVM.png → docs/images/MVVM_MVP.png**
  - Renamed the MVVM diagram to reflect its use in the MVP context.

New Files

- **docs/Architecture.md**
  - Adds the MVVM-based architecture overview, outlining the Data, Domain, and UI layers with descriptions and diagrams.
- **docs/ai_ready.md**
  - Documents the AI-ready commit message generation prompt and guidelines.
- **docs/images/MVVM_layers.png**
  - Introduces the foundational MVVM layer diagram asset.

- **docs/index.md**
  - Provides a project summary and table of contents for the documentation.
- **docs/introduction.md**
  - Adds the project introduction, MVP scope, and planned final version features.

Assets and Test Data

- **docs/images/MVVM_layers.png**
  - Foundational illustration of the MVVM layers.
- **docs/images/Diagrama_de_Classes.drawio**
  - Class diagram for the MVP implementation.

Conclusions

All documentation files have been added and existing resources reorganized under the `docs` directory; the system documentation is now complete and structured.

2025/07/08 save_product-02 - rudsonalves

Refactor column constants, DTO field mappings, and improve UI dialogs

This commit refactors the database column constants and SQL schema definitions to use abbreviated naming conventions, updates DTOs and models to support nullable descriptions and renamed JSON keys, enhances the shopping cart user case with comprehensive logging and error handling, corrects UI dialog naming and flow, and streamlines date formatting utilities.

Modified Files

- **lib/data/repositories/category/category_repository.dart**
  - Updated `orderBy` and `filter` parameters to reference `CatsColumns` and `SubCatsColumns` instead of the old `CategoriesColumns` and `SubCategoriesColumns`.
- **lib/data/services/database/database_service.dart**
  - Removed redundant null check for `id` in the `set` operation prior to database insert.
- **lib/data/services/database/tables/sql_tables.dart**

  - Renamed `CategoriesColumns` to `CatsColumns` and `SubCategoriesColumns` to `SubCatsColumns`.
  - Added new `name` and `priceVariation` fields to `ItemColumns`.
  - Updated table creation SQL to treat descriptions as nullable, enforce uniqueness on bar codes, and update foreign key references to use the new column constants.
  - Introduced detailed comments for each schema and added indexes for products, last price, categories, and sub-categories.

- **lib/domain/dto/cart_item_dto/cart_item_dto.dart**
  - Made `description` nullable.
  - Renamed fields `category` → `categoryName` and `sub_category` → `subCategoryName` with corresponding JSON key updates.

- **lib/domain/dto/product/product_dto.dart**
  - Made `description` nullable.
  - Renamed JSON keys and properties from `category` → `category_name` and `sub_category` → `sub_category_name`, adjusting factory and mapping logic.

- **lib/domain/enums/enums.dart**
  - Added new `ThreeState` enum with values `yes`, `no`, and `indeterminate` for dialog handling.

- **lib/domain/models/item/item_model.dart**
  - Annotated `priceVariation` with `@JsonKey(name: 'price_variation')` for proper JSON serialization.

- **lib/domain/models/product/product_model.dart**
  - Made `description` nullable.
  - Renamed model and JSON fields `category` → `categoryName` and `sub_category` → `subCategoryName`.

- **lib/domain/user_cases/shopping_cart_user_case.dart**
  - Improved initialization and save methods with `switch` cases for `Result`, logging successes and failures.
  - Renamed `save` to `saveItem` and updated call sites.

- **lib/ui/core/ui/dialogs/bottom_sheet_dialog.dart**
  - Corrected class name from `BottonSheetDialog` to `BottomSheetDialog` and updated state class accordingly.

- **lib/ui/core/ui/form_fields/date_form_field.dart**
  - Replaced `toDDMMYYYY()` with `toBrDate()` for Brazilian date formatting.

- **lib/ui/view/cart_shopping/add_product_cart/add_product_cart_view.dart**
  - Imported `ButtonSignature` and `BottomSheetDialog`.
  - Switched from `save` to `saving` command listener and merged listeners for state updates.
  - Enhanced barcode dialog flow using `ThreeState` responses.
  - Updated form field properties for capitalization, keyboard input, and null-safe descriptions.
  - Refactored `ListenableBuilder` to merge multiple notifiers.
  - Added snackbar feedback on save result and handled navigation pop on success.
- **lib/ui/view/cart_shopping/add_product_cart/add_product_cart_view_model.dart**
  - Renamed `save` command to `saving` and updated method call to `saveItem`.
  - Enhanced error logging logic in barcode lookup.
- **lib/ui/view/home/edit_shopping/edit_shopping_view.dart**
  - Added suffix icons and handlers to insert current date in name and description fields.
  - Imported `date_time_extensions` for utility methods.
- **lib/utils/extensions/date_time_extensions.dart**
  - Renamed `toDDMMYYYY` to `toBrDate` and updated `toBrDateTime` to use the new method.

Conclusions

All changes have been implemented and the application is fully functional.

2025/07/05 save_product-01 - rudsonalves

Add product_id and sale_by support to DTOs, models, and views

This commit enhances how cart items and pricing data are represented by introducing a nullable `product_id` field and a `saleBy` enum across DTOs, models, and view logic. It refactors factory constructors and mapping methods to streamline creation from `CartItemDto`, replaces the old `isUnitPrice` flag with a standardized enum, and updates dependency imports for consistency. View and ViewModel code now capture and propagate the product ID seamlessly.

Modified Files

- **docs/Diagrama_de_Classes.drawio**
  - Updated draw.io metadata to the latest version for compatibility with the current editor.

- **lib/config/dependencies.dart**
  - Corrected import paths for `cart_items` repository to use absolute project imports.
- **lib/domain/dto/cart_item_dto/cart_item_dto.dart**
  - Added `@JsonKey(name: 'product_id') String? productId` to the `CartItemDto` factory signature.
- **lib/domain/dto/last_price/last_price_dto.dart**
  - Removed the `isUnitPrice` boolean and `SqliteHelpers` import.
  - Introduced `@JsonKey(name: 'sale_by') required SaleBy saleBy` enum field.
- **lib/domain/dto/product/product_dto.dart**
  - Imported `CartItemDto`.
  - Renamed `isEqualModel` to `isEqualProductModel`.
  - Added `factory ProductDto.fromCartItemDto(CartItemDto dto)` to map directly from cart items.
- **lib/domain/models/item/item_model.dart**
  - Added `name` and `priceVariation` properties.
  - Refactored the `create` factory to include the new fields.
  - Introduced `factory ItemModel.fromCartItemDto(String? productId, CartItemDto dto)` for direct mapping.
- **lib/domain/models/last_price/last_price_model.dart**
  - Removed `isUnitPrice`, added `@JsonKey(name: 'sale_by') required SaleBy saleBy`.
  - Updated the `create` and DTO-to-model mapping to use the new enum.
- **lib/domain/models/product/product_model.dart**
  - Set `updatedAt: DateTime.now()` when mapping from DTO for consistency.
- **lib/ui/view/cart_shopping/add_product_cart/add_product_cart_view.dart**
  - Added a `_productId` field to hold the fetched product's ID.
  - Assigned `_productId` when a product is successfully loaded.
- **lib/ui/view/cart_shopping/add_product_cart/add_product_cart_view_model.dart**
  - Replaced manual `ProductDto` instantiation with `ProductDto.fromCartItemDto`.
  - Updated equivalence check to use `isEqualProductModel`.
  - Simplified item creation to `ItemModel.fromCartItemDto`.

Conclusions

All manual code and configuration changes are complete, and the application now fully supports product identification and standardized sale types across its data layer and UI.

2025/07/04 save_product - rudsonalves

Refactor imports to use absolute paths

This update standardizes all import statements across the project to use absolute package paths (`/…`) instead of relative or legacy `package:compras/...` forms, improving consistency and simplifying refactoring.

Modified Files

- **lib/config/dependencies.dart**
  - Swapped `package:compras/data/repositories/...` imports for absolute imports (`/data/repositories/...`).
- **lib/data/services/database/database_manager.dart**
  - Replaced domain model imports from `package:compras/...` to absolute imports (`/domain/models/...`).
- **lib/data/services/database/database_service.dart**
  - Updated exception and result utility imports to absolute paths (`/data/services/exceptions/...`, `/utils/result.dart`).
- **lib/domain/dto/last_price/last_price_dto.dart**
  - Changed SQLite helpers import to absolute path (`/domain/models/sqlite_helpers.dart`).
- **lib/domain/models/product/product_model.dart**
  - Reordered imports: moved `freezed_annotation` import above, and switched `enums.dart` import to absolute path (`/domain/enums/enums.dart`).
- **lib/domain/models/sub_category/sub_category_model.dart**
  - Converted DTO import to absolute path (`/domain/dto/sub_category/sub_category_dto.dart`).
- **lib/routing/router.dart**
  - Updated category repository and scanner view imports to absolute paths (`/data/repositories/category/i_category_repository.dart`, `/ui/view/scanner_barcode/scanner_barcode_view.dart`).
- **lib/ui/core/ui/form_fields/basic_form_field.dart**
  - Switched `dimens.dart` import to absolute path (`/ui/core/themes/dimens.dart`).
- **lib/ui/core/ui/form_fields/selection_field.dart**

  - Updated `BasicFormField` import to use absolute path (`/ui/core/ui/form_fields/basic_form_fi`

- **lib/ui/view/cart_shopping/add_product_cart/add_product_cart_view.dart**
  - Replaced domain model, fonts, and `SelectionField` imports with absolute paths.
- **lib/ui/view/cart_shopping/cart_shopping_view.dart**
  - Changed `routes.dart` import to absolute path (`/routing/routes.dart`).
- **lib/ui/view/cart_shopping/cart_shopping_view_model.dart**
  - Updated cart items repository import to absolute path (`/data/repositories/cart_items/i_cart_`
- **lib/ui/view/home/edit_shopping/edit_shopping_view.dart**
  - Added absolute import for `shopping_model.dart`.
- **lib/ui/view/home/edit_shopping/edit_shopping_view_model.dart**
  - Converted `result.dart` import to absolute path (`/utils/result.dart`).
- **lib/ui/view/home/widgets/shopping_list_tile.dart**
  - Updated multiple imports—model, fonts, dismissible container, and extensions—to absolute paths.

Conclusions

All import statements are now consistent and use absolute package paths, improving maintainability and reducing potential conflicts.

2025/07/04 last_price_repository-10 - rudsonalves

Introduce SelectionField and refine form components and cart UI

This commit replaces the deprecated suggestion text field with a new `SelectionField` widget, extends `BasicFormField` to support `onTap` callbacks, adjusts global dimension values, restructures the snack bar layout for consistent icon and title alignment, and enhances the AddProductCartView and its ViewModel to calculate and display the total dynamically.

Modified Files

- **lib/domain/user_cases/shopping_cart_user_case.dart**
  - Removed commented-out `subCategories` method stub that duplicated existing getter logic.
- **lib/ui/core/themes/dimens.dart**
  - Increased default `radius` value from `12.0` to `18.0` for all border radii.
- **lib/ui/core/ui/dialogs/app_snack_bar.dart**

  - Wrapped title and icon inside a centered `Row` when a title is present.
  - Moved the `Divider` to display below the title row.
  - Removed explicit icon size to rely on default sizing.

- **lib/ui/core/ui/form_fields/basic_form_field.dart**
  - Added new `onTap` callback parameter to the widget API.
  - Hooked `onTap` into the underlying text field to handle read-only interactions.

- **lib/ui/view/cart_shopping/add_product_cart/add_product_cart_view.dart**
  - Swapped out `SugestionTextField` for the new `SelectionField` in category and sub-category inputs.
  - Imported `selection_field.dart` and `fonts.dart` instead of the deleted suggestion field.
  - Introduced a `_total` `ValueNotifier<double>` and listeners on price, quantity, and weight to compute the line total.
  - Added a `ValueListenableBuilder` to display the formatted total in the UI.
  - Updated the "Add" button icon from `Symbols.add_rounded` to `Symbols.add_shopping_cart_rounded`.
  - Refactored row layout for categories and sub-categories into two `Expanded` `SelectionField` widgets.
  - Simplified form validation logic in `_saving()` by correcting the conditional check.

- **lib/ui/view/cart_shopping/add_product_cart/add_product_cart_view_model.dart**
  - Standardized import paths to use absolute package references.
  - Updated `subCategoriesNames` to filter by `categoryId` before mapping names.

- **lib/ui/core/ui/form_fields/sugestion_text_field.dart**
  - Deleted obsolete `SugestionTextField` implementation now superseded by `SelectionField`.

New Files

- **lib/ui/core/ui/form_fields/selection_field.dart** Provides a read-only dropdown-like text field with built-in menu options for selecting from a list of suggestions and handling focus transitions.

Conclusions

All form components and cart UI enhancements are complete and functional.

2025/07/04 last_price_repository-09 - rudsonalves

Refactor repo structure, add CartItemDto and SuggestionTextField

This commit reorganizes the cart items repository paths, enriches category and product repository interfaces and implementations, and introduces robust error handling in the database layer. It also adds a new `CartItemDto` for decoupling item creation from domain models, enhances the `ProductDto` and `ProductModel` to support category metadata, and implements a `SuggestionTextField` component for improved category/subcategory selection in the UI.

Modified Files

- **docs/Diagrama_de_Classes.drawio**
  - Updated Draw.io file to the latest version metadata.
- **lib/config/dependencies.dart**
  - Adjusted import paths for `cart_items` repositories.
  - Initialized `CategoryRepository` before providing it.
  - Replaced inline provider creation with the pre-initialized `categoryRepository` instance.
- **lib/data/repositories/cart_items/**
  - Renamed `items/cart_items_repository.dart` and `items/i_cart_items_repository.dart` to `cart_items/cart_items_repository.dart` and `cart_items/i_cart_items_repository.dart`.
- **lib/data/repositories/category/category_repository.dart**
  - Renamed `fetchCategories` → `fetchAllCategories` and `fetchSubCategories` → `fetchAllSubCategories`.
  - Removed single `_categoryId` field in favor of a `_loadsCategoryIds` cache list.
  - Added `category(String)` getter, `getSubCategory(...)` async lookup, and private helper `_subCategoriesList(...)`.
  - Updated `updateSubCategory` signature casing.
  - Streamlined subcategory caching logic.
- **lib/data/repositories/category/i_category_repository.dart**
  - Updated interface to match renamed methods: `fetchAllCategories`, `fetchAllSubCategories`.
  - Added getters `subCategories`, `getSubCategory(...)`, and `category(String)`.
- **lib/data/repositories/products/i_products_repository.dart**
  - Added `fetchByBarCode(String barCode)` to interface.
- **lib/data/repositories/products/products_repository.dart**

  - Implemented `fetchByBarCode` with filter query and in-memory caching.

- **lib/data/services/database/database_service.dart**
  - Imported `RecordNotFoundException`.
  - Replaced all generic `Exception('No record found…')` throws with `RecordNotFoundException` for consistency.

- **lib/data/services/database/tables/sql_tables.dart**
  - Added `categoryName` and `subCategoryName` columns to `ProductColumns` and SQL table schema.

- **lib/domain/dto/product/product_dto.dart**
  - Imported `ProductModel`.
  - Added `category` and `subCategory` fields.
  - Introduced `isEqualModel` method to compare DTO and model.

- **lib/domain/models/product/product_model.dart**
  - Added `category` and `subCategory` fields and updated JSON serialization in `.g.dart`.

- **lib/domain/user_cases/shopping_cart_user_case.dart**
  - Updated imports to new `cart_items` path.
  - Exposed `categories` and `subCategories` getters.
  - Implemented `findProductByBarCode` to fetch the product and preload subcategories.
  - Renamed and added helper methods for subcategory fetching.

- **lib/routing/router.dart**
  - Updated import for cart items repository.
  - Changed AddProductCart route builder to accept `ShoppingModel` via `state.extra`.

- **lib/ui/core/ui/form_fields/basic_form_field.dart**
  - Added `errorText`, `labelStyle`, `onFieldSubmitted`, `initialValue` parameters.
  - Wired `initialValue`, `errorText`, `labelStyle` into `InputDecoration` and field callbacks.

- **lib/ui/view/cart_shopping/add_product_cart/add_product_cart_view.dart**
  - Switched to `SugestionTextField` for category/subcategory input.
  - Added `ShoppingModel` to widget constructor and route passing.
  - Introduced `RecordNotFoundException`–based UI messaging.
  - Updated save logic to build `CartItemDto`.

- **lib/ui/view/cart_shopping/add_product_cart/add_product_cart_view_model.dart**

  - Switched save command to accept `CartItemDto`.
  - Added `fetchAllSubcategories` command.
  - Implemented product creation/update before item save.
- **lib/ui/view/cart_shopping/cart_shopping_view_model.dart**
  - Updated import path for cart items repository.

New Files

- **lib/data/services/exceptions/exceptions.dart** — Defines `RecordNotFoundException` for clearer error handling.
- **lib/domain/dto/cart_item_dto/cart_item_dto.dart** — Freezed DTO for cart item creation.
- **lib/ui/core/ui/form_fields/sugestion_text_field.dart** — New auto-complete text field for suggestion-based input.

Conclusions

All changes are complete and the system remains fully functional.

2025/07/03 last_price_repository-08 - by rudsonalves

Enhance category repo, add QR scanner flow, and rename number controller

This update improves subcategory loading by ID, integrates a QR code scanner into the "Add Product" workflow, and standardizes numeric input handling. It refactors product DTO/model fields to use `category_id` and `sub_category_id`, injects new commands into the view model, and adds utility and scanner view files.

Modified Files

- **Makefile**
  - Added `run_build_runner_watch` target and swapped its behavior with `run_build_runner`.
- **lib/config/dependencies.dart**
  - Registered `ICategoryRepository` with `CategoryRepository(dbService)`.
- **lib/data/repositories/category/category_repository.dart**
  - Introduced `_loadsCategoryIds` to avoid redundant subcategory fetches.
  - Removed unnecessary clearing of `_subCategories`.
  - Added `fetchSubCategory(String)` to load a single subcategory by ID.

  - Moved and documented `_findCategoryId` and `_findSubCategoryId` helper methods.

- **lib/data/repositories/category/i_category_repository.dart**
  - Extended interface with `fetchSubCategory` declaration and detailed DartDoc for each method.

- **lib/domain/dto/product/product_dto.dart**
  - Renamed `category` → `@JsonKey('category_id') categoryId` and `subcategory` → `@JsonKey('sub_category_id') subCategoryId`.

- **lib/domain/models/product/product_model.dart**
  - Updated model fields to match DTO: `categoryId` and `subCategoryId`.

- **lib/domain/user_cases/shopping_cart_user_case.dart**
  - Injected `ICategoryRepository`.
  - Added `findProductByBarCode` and `getSubCategory` methods to support lookup by barcode and subcategory retrieval.

- **lib/routing/router.dart**
  - Provided `categoryRepository` to `ShoppingCartUserCase`.
  - Added `GoRoute` for the scanner view.
  - Imported `QrCodeScannerView`.

- **lib/routing/routes.dart**
  - Declared new `scanner` route constant.

- **lib/ui/core/ui/editing_controllers/number_editing_controller.dart**
  - Renamed from `currency_editing_controller.dart` to `number_editing_controller.dart`.
  - Changed regex to allow digits only and renamed `currencyValue` → `numberValue`.

- **lib/ui/view/cart_shopping/add_product_cart/add_product_cart_view.dart**
  - Integrated snack bar feedback and GoRouter push to the scanner route.
  - Added listener for `findProductByBarCode`, showing a loading dialog and populating fields on success.
  - Captured scanned code and triggered barcode lookup.
  - Stored `categoryId`/`subCategoryId` for subsequent subcategory fetch.

- **lib/ui/view/cart_shopping/add_product_cart/add_product_cart_view_model.dart**
  - Added `findProductByBarCode` and `getSubCategory` `Command1` instances.

New Files

- **lib/ui/view/scanner_barcode/scanner_barcode_view.dart** Implements a full-screen QR/barcode scanner using `mobile_scanner`, returning the scanned value on pop.
- **lib/utils/extensions/list_equality.dart** Provides `ListEquality` for deep list comparisons, copied from the Dart `collection` package.

Conclusion

With these changes, the app now supports barcode-driven product lookup, efficient subcategory fetching, and a unified numeric input controller, ensuring a smoother "Add Product" experience.

2025/07/02 last_price_repository-07 - by rudsonalves

Implement category management layer and enrich DatabaseService documentation

This commit introduces a new category management layer by defining the `ICategoryRepository` interface, its `CategoryRepository` implementation, and the `SubCategoryDto`. It also registers the category repository in the dependency graph and enriches the `DatabaseService` with detailed documentation for common CRUD operations.

Modified Files

- **Makefile**
  - Added `run_build_runner_watch` target for selective build_runner watching.
  - Swapped the definitions of `run_build_runner_watch` and `run_build_runner` commands.
- **lib/config/dependencies.dart**
  - Imported `ICategoryRepository` and `CategoryRepository`.
  - Provided `CategoryRepository(dbService)` under `ICategoryRepository` in the dependency list.
- **lib/data/services/database/database_service.dart**
  - Added XML-style DartDoc comments for `fetchById`, `fetchByFilter`, `fetchAll`, `insert`, `set`, `update`, `updateWhere`, `delete`, and `deleteWhere`.
  - Clarified parameter descriptions and return semantics for each CRUD method.
- **lib/domain/dto/product/product_dto.dart**

  - Adjusted import to use project-root relative path for `SaleBy` enum (`/domain/enums/enums.dart`).

- **lib/domain/models/sub_category/sub_category_model.dart**
  - Imported `SubCategoryDto`.
  - Added `factory SubCategoryModel.fromDto(String id, SubCategoryDto dto)` for DTO-to-model mapping.

New Files

- **lib/data/repositories/category/i_category_repository.dart** Defines the `ICategoryRepository` interface for fetching, inserting, updating, and deleting categories and subcategories.
- **lib/data/repositories/category/category_repository.dart** Implements `ICategoryRepository`, managing in-memory caches and delegating CRUD calls to `DatabaseService`.
- **lib/domain/dto/sub_category/sub_category_dto.dart** Freezed DTO class for serializing and deserializing subcategory payloads, with JSON support.

Conclusion

Category management infrastructure is now in place, and the `DatabaseService` is fully documented for robust data operations.

2025/07/02 last_price_repository-06 - by rudsonalves

Implement category support, saleBy enums, and enhance cart flow

This commit introduces full support for product categories and subcategories across the database, DTOs, models, and UI. It also replaces the deprecated `isUnitPrice` flag with a `SaleBy` enum, updates routing for the cart flow, and enriches the "Add to Cart" form with quantity, weight, and category selectors. Additionally, the database bootstrap now populates initial category data.

Modified Files

- **Makefile**
  - Refined `run_build_runner` command to only watch domain models and DTOs under `lib/domain/models/**` and `lib/domain/dtos/**`.
- **docs/Diagrama_de_Classes.drawio**
  - Updated class diagram to reflect new `CategoryModel`, `SubCategoryModel`, and `SaleBy` relationships.
- **ios/Runner/Info.plist**

  - Added `NSCameraUsageDescription` and `NSPhotoLibraryUsageDescription` keys for QR code scanning and photo library access.

- **lib/data/repositories/items/cart_items_repository.dart**
  - Removed obsolete `_isInitialized` flag.
  - Changed initialization guard to check `_shoppingId` equality instead of a boolean.

- **lib/data/services/database/database_manager.dart**
  - Imported `CategoryModel`, `SubCategoryModel`, and `Uuid`.
  - Created `categories` and `subCategories` tables and indexes in `_onCreate`.
  - Added `_populateCategoriesTables` to seed initial category/subcategory data on database creation.

- **lib/data/services/database/tables/sql_configurations.dart**
  - Defined `categories` map constant with all category names and their subcategories.

- **lib/data/services/database/tables/sql_tables.dart**
  - Added `Tables.categories`, `Tables.subCategories`, `CategoriesColumns`, and `SubCategoriesColumns`.
  - Modified `products`, `items`, and `lastPrice` table definitions to include `sale_by`, `category_id`, and `sub_category_id` columns, plus corresponding foreign keys.
  - Added creation statements and indexes for `categories` and `sub_categories` tables.

- **lib/domain/dto/product/product_dto.dart**
  - Replaced `isUnitPrice` boolean with `saleBy: SaleBy` enum.
  - Added optional `category` and `subcategory` fields.

- **lib/domain/enums/enums.dart**
  - Renamed `byUnit`/`byWeight` to `unit`/`weight` in `SaleBy` enum, updating their labels.

- **lib/domain/user_cases/shopping_cart_user_case.dart**
  - Added `save` and `update` stubs returning delayed `Result.success`, preparing for real repository calls.

- **lib/routing/router.dart**
  - Switched `GoRoute` for `/shopping` into a `ShellRoute`, providing `ShoppingCartUserCase` via `Provider`.
  - Updated nested routes for `addProductCart`, passing `userCase` to view models.

- **lib/routing/routes.dart**
  - Adjusted `addProductCart` route's path to be relative under `/shopping`.
- **lib/ui/core/ui/form_fields/basic_form_field.dart**
  - Added `prefixIcon`, `suffixIcon`, `prefixText`, `suffixText`, and `textAlign` properties.
  - Defaulted to `widget.border` or an `OutlineInputBorder` from theme.
  - Unified icon logic to respect provided widgets over icons.
- **lib/ui/view/cart_shopping/add_product_cart/add_product_cart_view.dart**
  - Integrated `_barCodeController`, category and subcategory text fields.
  - Introduced `ValueNotifier<SaleBy>` to switch between unit/weight modes.
  - Added quantity increment/decrement and weight input logic.
  - Wrapped action button in `ListenableBuilder` and replaced with `BigButton`.
  - Disposed all controllers in `dispose()`.
- **lib/ui/view/cart_shopping/add_product_cart/add_product_cart_view_model.dart**
  - Injected `ShoppingCartUserCase` dependency.
  - Defined `save` and `update` commands invoking user case methods.
- **lib/ui/view/cart_shopping/cart_shopping_view.dart**
  - Updated `_addProductCart()` to push nested route including shopping path and extra model.
- **lib/utils/validates/generic_validations.dart**
  - Added `notZero` validator to ensure numeric inputs are greater than zero.
- **pubspec.yaml**
  - Bumped `go_router` to `^16.0.0`.
  - Added `mobile_scanner: ^7.0.1` dependency for barcode scanning.

New Files

- **lib/domain/models/category/category_model.dart** Defines the `CategoryModel` Freezed data class for product categories.
- **lib/domain/models/sub_category/sub_category_model.dart** Defines the `SubCategoryModel` Freezed data class, linking to `CategoryModel`.

Conclusion

With these changes, category management and the enhanced cart workflow—including sale modes, routing, and form improvements—are fully integrated and functional.

2025/07/01 last_price_repository-05 - by rudsonalves

Refactor cart dependencies and implement AddProductCart feature

This commit refactors dependency imports and the shopping cart flow to support a new "Add Product to Cart" feature. It standardizes import paths, replaces the cart items provider with `ChangeNotifierProvider`, and renames existing shopping views and view models to the cart-specific equivalents. A new screen and view model for adding products to the cart have also been added.

Modified Files

`lib/config/dependencies.dart`

- Switched relative imports to absolute imports for repository interfaces and implementations.
- Replaced `Provider<ICartItemsRepository>` with `ChangeNotifierProvider<ICartItemsRepository>` for cart items.

`lib/domain/enums/enums.dart`

- Introduced a new `SaleBy` enum with two values (`byUnit` and `byWeight`) and associated labels.

`lib/routing/router.dart`

- Updated imports to include `CartShoppingView`, `CartShoppingViewModel`, `AddProductCartView`, and `AddProductCartViewModel`.
- Renamed the shopping route builder to return `CartShoppingView` with `CartShoppingViewModel`.
- Added a new `GoRoute` for the `/add-product-cart` path.

`lib/routing/routes.dart`

- Added a new `addProductCart` constant route.

`lib/ui/view/cart_shopping/cart_shopping_view.dart`

- Renamed `ShoppingView` to `CartShoppingView`.
- Updated constructor and state class names accordingly.
- Changed the floating action button's callback to navigate to the new add-product-cart route.

`lib/ui/view/cart_shopping/cart_shopping_view_model.dart`

- Renamed `ShoppingViewModel` to `CartShoppingViewModel` and updated constructor to match the new class name.

New Files

`lib/ui/view/cart_shopping/add_product_cart/add_product_cart_view.dart`
Defines a stateful form view for adding a product to the cart, including fields for name, description, sale type, price, and quantity, plus a barcode scanner button placeholder.

`lib/ui/view/cart_shopping/add_product_cart/add_product_cart_view_model.dart`
Introduces an empty `AddProductCartViewModel` class as the view model for the add-product-cart screen.

Conclusion

All cart-related dependencies and views have been updated, and the Add Product to Cart feature is now integrated and ready for further development.

2025/06/30 last_price_repository-04 - by rudsonalves

Rename item repository to cart repository and integrate shopping cart user case

This commit refactors the item repository into a cart-specific implementation with ChangeNotifier support, updates dependency injection and routing to use the new `ShoppingCartUserCase`, and enhances the edit and shopping views to handle both create and update workflows. The `ShoppingModel` default `totalPrice` is adjusted to 0, and the class diagram is updated to reflect the renamed components. UI and view models are wired to the new use case for a cohesive shopping cart flow.

Alterations

- **docs/Diagrama_de_Classes.drawio**
  - Update mxGraphModel `dx` and `dy` values for layout adjustment.
  - Rename diagram shapes: `ItemsRepository` → `CartItemsRepository` and `ShoppingUserCase` → `ShoppingCartUserCase`.
- **lib/config/dependencies.dart**
  - Replace imports of `i_items_repository.dart`/`items_repository.dart` with `i_cart_items_repository.dart`/`cart_items_repository.dart`.
  - Update provider registration to use `ICartItemsRepository` and `CartItemsRepository`.

- **lib/data/repositories/items/cart_items_repository.dart** (formerly `items_repository.dart`)
  - Rename class to `CartItemsRepository`, implement `ICartItemsRepository`, and mix in `ChangeNotifier`.
  - Add `notifyListeners()` calls after each mutation of `_items`.
  - Update constructor signature to `CartItemsRepository`.
- **lib/data/repositories/items/i_cart_items_repository.dart** (formerly `i_items_repository.dart`)
  - Rename abstract class to `ICartItemsRepository` extending `ChangeNotifier`.
- **lib/data/repositories/shopping/shopping_repository.dart**
  - Mix in `ChangeNotifier` on `ShoppingRepository`.
  - Import `package:flutter/material.dart` to support listener notifications.
- **lib/domain/models/shopping/shopping_model.dart**
  - Change `totalPrice` to use `@Default(0)` and remove `required`.
  - Provide default value `0` in the factory constructor.
- **lib/routing/router.dart**
  - Add imports for `ICartItemsRepository`, `IProductsRepository`, `ILastPriceRepository`, and `ShoppingCartUserCase`.
  - Modify `EditShopping` route builder to accept a `ShoppingModel` via `state.extra`.
  - Wire `ShoppingCartUserCase` into `ShoppingViewModel` builder for the shopping route.
- **lib/ui/view/home/edit_shopping/edit_shopping_view.dart**
  - Import `ShoppingModel` and add optional `shopping` parameter to `EditShoppingView`.
  - Introduce `_isEditing` flag and `_initializeForm()` to prefill fields when editing.
  - Adjust AppBar title, button label, and icon based on mode.
  - Listen to both `saving` and new `update` commands and execute update flow when editing.
- **lib/ui/view/home/edit_shopping/edit_shopping_view_model.dart**
  - Add `update Command1<ShoppingModel, ShoppingModel>` for record updates.
  - Implement `_update()` method calling `shoppingRepository.update()` and logging the result.
- **lib/ui/view/home/home_view.dart**

  - Rename `_newShopping()` to `_addShopping()` and update `floatingActionButton.onPressed`.
  - Enable edit navigation by calling `context.push(Routes.editShopping.path, extra: shopping)` in `_editShopping()`.

- **lib/ui/view/home/shopping/shopping_view.dart**
  - Add a `FloatingActionButton` placeholder for future cart actions.
  - Wrap body in `ListenableBuilder` listening to `viewModel.cartNotifier` to react to cart changes.

- **lib/ui/view/home/shopping/shopping_view_model.dart**
  - Implement `ShoppingViewModel` to depend on `ShoppingCartUserCase`.
  - Initialize `load` command to invoke `userCase.load()` and expose `cartNotifier`.

New Files

- **lib/domain/user_cases/shopping_cart_user_case.dart** Introduce `ShoppingCartUserCase` to coordinate initialization of product, cart items, and last price repositories for a given `ShoppingModel`.

Conclusion

The refactor is complete and the shopping cart flow is now fully functional.

2025/06/28 last_price_repository-03 - by rudsonalves

Refactor Shopping flow to use IShoppingRepository and introduce edit route

This commit abstracts `ShoppingRepository` behind the `IShoppingRepository` interface, updates DI and routing to use the new interface type, and renames the "new shopping" feature into a unified edit flow with dedicated views and view models. Navigation paths and provider registrations now reflect the interface contract.

Modified Files

- **lib/config/dependencies.dart**
  - Imported `IShoppingRepository` and registered it with `ChangeNotifierProvider<IShoppingRepos` instead of concrete `ShoppingRepository`.

- **lib/data/repositories/shopping/i_shopping_repository.dart**
  - Extended `ChangeNotifier` in `IShoppingRepository` to support UI listeners.

- **lib/data/repositories/shopping/shopping_repository.dart**

  - Removed `ChangeNotifier` mixin and `implements`; now extends `IShoppingRepository`.

- **lib/routing/router.dart**
  - Swapped `ShoppingRepository` references for `IShoppingRepository` in builders.
  - Renamed `newShopping` route to `editShopping` and added a `shopping` detail route.
  - Adjusted route imports and added `ShoppingView` entry.

- **lib/routing/routes.dart**
  - Renamed `new_shopping` to `editShopping` (`/edit-shopping`) and added `shopping` (`/shopping`) path.

- **lib/ui/view/home/home_view.dart**
  - Tweaked import order.
  - Updated `_newShopping` to push `editShopping` route.
  - Updated `_openShopping` to push new `shopping` route with extra model.

- **lib/ui/view/home/home_view_model.dart**
  - Changed type of `_shoppingRepository` parameter from concrete to `IShoppingRepository`.

New Files

- **lib/ui/view/home/edit_shopping/edit_shopping_view.dart** Renamed and refactored from `new_shopping.dart` to handle both creation and editing flows.
- **lib/ui/view/home/edit_shopping/edit_shopping_view_model.dart** Renamed view model to `EditShoppingViewModel` matching the updated view.
- **lib/ui/view/home/shopping/shopping_view.dart** New detail screen displaying a single `ShoppingModel`.
- **lib/ui/view/home/shopping/shopping_view_model.dart** Placeholder view model for the new `ShoppingView`.

Conclusion

The shopping creation and editing flows now share a unified interface via `IShoppingRepository`, and routing has been updated to support both listing and detail views. The refactor aligns DI, routing, and UI layers under the new interface abstraction.

2025/06/28 last_price_repository-02 - by rudsonalves

Introduce DatabaseManager, DI updates, and refactor DatabaseService

This commit centralizes database initialization into a new `DatabaseManager` class, refactors `DatabaseService` to consume a managed `Database` instance, and updates dependency injection to provide the new manager and service. Repository providers and integration tests are adjusted accordingly.

Modified Files

- **lib/config/dependencies.dart**
  - Replaced direct `DatabaseService.initialize` with `DatabaseManager` and injected both manager and service.
  - Registered `IProductsRepository`, `IItemsRepository`, and `ILastPriceRepository` alongside existing `ShoppingRepository`.
- **lib/data/services/database/database_service.dart**
  - Removed internal initialization logic; now accepts a pre-initialized `Database` instance.
  - All `_db` references replaced with the injected `_database` field.
- **lib/data/services/database/tables/sql_tables.dart**
  - Added foreign key constraints to `items` and `lastPrices` tables for cascading deletes.
- **lib/domain/models/last_price/last_price_model.dart**
  - Corrected import order to prioritize DTO and helper imports after annotations.
- **test/data/repositories/products/products_repository_test.dart**
  - Switched to use `DatabaseManager` for setup and teardown instead of manual `DatabaseService` init/close.
- **test/data/repositories/shopping/shopping_repository_test.dart**
  - Updated to initialize `DatabaseService` via `DatabaseManager` and close manager in `tearDown`.

New Files

- **lib/data/services/database/database_manager.dart** Manages SQLite database lifecycle: creation, configuration (foreign keys), upgrades/downgrades, and closing.
- **lib/data/services/database/tables/sql_configurations.dart** Defines database name, version, and PRAGMA statements for configuration.

Conclusion

Database initialization is now unified under `DatabaseManager`, DI graph updated, and tests adapted—ensuring consistent database handling across the app.

2025/06/27 last_price_repository-01 - by rudsonalves

Extend SQLite boolean handling and add LastPrice repository/tests

This commit refines SQLite integer-to-boolean conversion via a new `SqliteHelpers` utility, updates schema columns for `is_unit_price` fields, and integrates a full-featured LastPrice repository with corresponding interfaces and unit tests. Repository interfaces for items now include `totalPrice()`, and the shared `Result` class gains success/failure flags.

Modified Files

- **docs/Diagrama_de_Classes.drawio**
  - Updated internal diagram dimensions to match the latest class revisions.
- **docs/images/Diagrama_de_Classes.png**, **docs/images/MVVM.png**
  - Replaced with refreshed exports reflecting updated column and model changes.
- **lib/data/repositories/items/i_items_repository.dart**
  - Added `totalPrice()` signature to the items repository interface.
- **lib/data/repositories/items/items_repository.dart**
  - Implemented `@override totalPrice()` annotation for consistency.
- **lib/data/services/database/tables/sql_tables.dart**
  - Changed DDL default types for `is_unit_price` from BOOLEAN to INTEGER (0/1).
- **lib/domain/dto/last_price/last_price_dto.dart**
  - Introduced `fromJson`/`toJson` converters (`SqliteHelpers`) on `isUnitPrice` key.
- **lib/domain/dto/product/product_dto.dart**
  - Applied same integer-to-boolean converters to `ProductDto.isUnitPrice`.
- **lib/domain/models/item/item_model.dart**
  - Added JSON converters on `ItemModel.isUnitPrice`.
- **lib/domain/models/last_price/last_price_model.dart**

  - Applied converters to `LastPriceModel.isUnitPrice`.

- **lib/domain/models/product/product_model.dart**
  - Integrated converters on `ProductModel.isUnitPrice`.

- **lib/utils/result.dart**
  - Introduced `isSuccess` and `isFailure` getters to simplify result checking.

- **pubspec.yaml**
  - Added `sqflite_common_ffi` dependency for desktop testing and `path_provider`.

New Files

- **lib/domain/models/sqlite_helpers.dart** Utility class for converting between Dart `bool` and SQLite integer (0/1).
- **lib/data/repositories/last_price/i_last_price_repository.dart** Repository interface defining CRUD operations for last-price entries.
- **lib/data/repositories/last_price/last_price_repository.dart** Concrete implementation managing in-memory cache and database persistence of last-price records.
- **test/data/repositories/products/products_repository_test.dart** Integration tests for `ProductsRepository` using in-memory SQLite via `sqflite_common_ffi`.
- **test/data/repositories/shopping/shopping_repository_test.dart** Integration tests for `ShoppingRepository` with desktop SQLite support.

Conclusion

All changes for integer-based boolean handling and the new LastPrice feature are in place, and comprehensive tests confirm full functionality.

2025/06/27 structural_adjustments-01 - by rudsonalves

Add last-price tracking and unit-price support across data models

This commit extends the shopping app's data model and persistence layer to support "last price" records and unify unit-based pricing. Schema definitions, repositories, DTOs, and domain models have been updated accordingly, and documentation (README and class diagrams) reflects these enhancements.

Modified Files

- **README.md**

  - Expanded project description with MVVM architecture overview.
  - Added class- and MVVM-diagram image references.
  - Clarified three-layer structure (Data, Domain, UI) and main architectural branches.

- **docs/Diagrama_de_Classes.drawio**
  - Adjusted `mxGraphModel` dimensions to match updated diagrams.

- **lib/data/repositories/items/items_repository.dart**
  - Renamed `prince` → `unitPrice` field.
  - Updated `totalPrice()` to use `unitPrice` instead of legacy `prince`.

- **lib/data/services/database/database_service.dart**
  - Included execution of `lastPrices` table and its index in batch setup.
  - Added grouping comments for "Tables" and "Indexes".

- **lib/data/services/database/tables/sql_tables.dart**
  - Added `last_price` table DDL and index.
  - Introduced `is_unit_price`, `unit_price`, and `created_at` columns for `items`.
  - Added `is_unit_price` flag to `products`.

- **lib/domain/dto/product/product_dto.dart**
  - Added `isUnitPrice` boolean (default `true`) to `ProductDto` fields and factory.

New Files

- **docs/images/Diagrama_de_Classes.png** PNG export of the updated class diagram.
- **docs/images/MVVM.png** Visualization of the MVVM architecture branches.
- **lib/domain/dto/last_price/last_price_dto.dart** Freezed DTO for "last price" records, including `productId`, `isUnitPrice`, `lastUnitPrice`, and `createdAt`.
- **lib/domain/models/last_price/last_price_model.dart** Domain model for "last price" entries with factory constructors and JSON support.

Assets and Test Data

- **Diagram Images**
  - `docs/images/Diagrama_de_Classes.png`
  - `docs/images/MVVM.png`

Conclusion

All enhancements for last-price tracking and unit-based pricing are in place, and the application is now fully functional.

2025/06/27 home_view-test-01 - by rudsonalves

Update class diagram, dependency injection, repositories, DTOs, and UI enhancements

This commit refines the class diagram geometry, modernizes dependency registration to use `ChangeNotifierProvider`, enriches repositories with notification support, adds default values and nullability to DTOs, introduces rich UI interactions in the Home view (including swipe-to-edit/delete and a shopping list tile), embeds new enum icons, and extends the diagram with a `LastPriceModel` swimlane.

Modified Files

- **docs/Diagrama_de_Classes.drawio**
  - adjusted `dx`/`dy` values, swimlane geometries, and edge point arrays
  - highlighted `ShoppingDto`/`ProductDto` parameters in bold
  - added `LastPriceModel` swimlane with fields and repository edges
- **lib/config/dependencies.dart**
  - replaced dual `Provider<IShoppingRepository>` with a single `ChangeNotifierProvider<ShoppingRepository>`
  - commented out legacy interface provider
- **lib/data/repositories/shopping/shopping_repository.dart**
  - extended `ShoppingRepository` from `ChangeNotifier`
  - added `notifyListeners()` on insert, update, delete, and load
  - imported `flutter/material.dart` to support `ChangeNotifier`
- **lib/domain/dto/shopping/shopping_dto.dart**
  - moved `totalPrice` default into constructor, removed explicit `required`
- **lib/domain/enums/enums.dart**
  - added `IconData iconData` to `ShoppingType` enum entries
  - imported `material_symbols_icons`
- **lib/routing/router.dart**
  - reordered imports, read repository via `ctx.read<ShoppingRepository>()`
  - aligned builder contexts (`ctx` vs `context`)
- **lib/ui/view/home/home_view.dart**

  - wired `HomeViewModel.delete` listener for delete feedback
  - wrapped body in `ListenableBuilder` showing loading, empty state, or `ListView`
  - implemented swipe-to-edit and swipe-to-delete via `Dismissible`

- **lib/ui/view/home/home_view_model.dart**
  - switched from interface to `ShoppingRepository` dependency
  - initialized `load` immediately and added `delete` command
  - exposed `shoppings` and `notifier` from repository

- **lib/ui/view/new_shoppint/new_shopping.dart**
  - switched to `ShoppingDto.create(...)` factory

- **lib/ui/view/new_shoppint/new_shopping_view_model.dart**
  - removed artificial 2 s delay after save

- **lib/domain/dto/shopping/shopping_dto.freezed.dart** and **.g.dart**
  - no manual edits (regenerated by build runner)

New Files

- **lib/ui/view/home/widgets/shopping_list_tile.dart** Stateless widget displaying each shopping entry with leading icon, title, subtitle, formatted date, price label, and swipe actions for edit/delete.

Conclusion

All components are now reactive with provider notifications, DTOs are safer with defaults and nullability, and the UI supports robust list interactions—system is ready for further feature development.

2025/06/26 home_view-04 - by rudsonalves

Add nullable timestamps, default values, NewShopping feature, and build runner command

These changes enhance data handling by making `createdAt`/`updatedAt` fields nullable and providing default values, streamline development with a new `run_build_runner` target, and introduce a complete "New Shopping" flow. UI components, routing, and theming adjustments ensure consistency and usability across the app.

Modified Files

- **Makefile**
  - appended a `run_build_runner` target to watch code generation

- **docs/Diagrama_de_Classes.drawio**
  - adjusted geometry coordinates and updated swimlane labels for consistency
- **lib/config/dependencies.dart**
  - removed duplicate `IShoppingRepository` provider registration
- **lib/domain/dto/product/product_dto.dart**
  - changed `createdAt`/`updatedAt` to nullable
- **lib/domain/dto/product/product_dto.freezed.dart**
  - updated copyWith and constructor signatures to accept nullable timestamps
- **lib/domain/dto/product/product_dto.g.dart**
  - added null checks when parsing/serializing JSON dates
- **lib/domain/dto/shopping/shopping_dto.dart**
  - introduced default `totalPrice` of 0 and made timestamps nullable
- **lib/domain/dto/shopping/shopping_dto.freezed.dart**
  - adjusted generated code for default value and nullable fields
- **lib/domain/dto/shopping/shopping_dto.g.dart**
  - support null JSON timestamps and default `total_price`
- **lib/domain/models/product/product_model.dart**
  - made `createdAt`/`updatedAt` nullable
- **lib/domain/models/product/product_model.freezed.dart**
  - updated generated code to handle nullable timestamps
- **lib/domain/models/product/product_model.g.dart**
  - added null checks in JSON parsing/serialization
- **lib/domain/models/shopping/shopping_model.dart**
  - made `createdAt`/`updatedAt` nullable
- **lib/domain/models/shopping/shopping_model.freezed.dart**
  - updated generated code for nullable timestamps
- **lib/domain/models/shopping/shopping_model.g.dart**
  - handle optional JSON date fields
- **lib/routing/router.dart**

  - imported and registered `NewShopping` route

- **lib/routing/routes.dart**
  - added `newShopping` route constant
- **lib/ui/core/themes/dimens.dart**
  - increased `spacingVertical` from 12.0 to 18.0
- **lib/ui/core/ui/buttons/big_button.dart**
  - commented out unused `colorScheme` reference
- **lib/ui/core/ui/form_fields/basic_form_field.dart**
  - removed unused import and commented out default border logic
- **lib/ui/core/ui/form_fields/enum_form_field.dart**
  - introduced `EnumFormLayout` enum and added layout–dependent rendering (`Wrap`, `Row`)
- **lib/ui/view/home/home_view.dart**
  - simplified new-shopping action to use `context.push` with GoRouter

New Files

- **lib/ui/view/new_shoppint/new_shopping.dart** Provides the UI scaffolding and form for creating a new shopping entry, including validation and feedback.
- **lib/ui/view/new_shoppint/new_shopping_view_model.dart** Implements the view model for shopping creation, orchestrating repository calls and handling success/error logging.

Conclusion

All changes implemented successfully and the New Shopping feature is fully functional.

2025/06/26 home_view-03 - by rudsonalves

Add ItemsRepository and item table support; unify database APIs and adjust repository imports

Introduction

This commit introduces a new `IItemsRepository` interface and its `ItemsRepository` implementation to manage shopping items. The database service gains `fetchByFilter`, `updateWhere`, and `deleteWhere` methods; SQL table definitions now include an `items` table with appropriate columns and primary keys. Repository constructors are standardized to use a single

`_dbService` instance, and import ordering is aligned. Freezed annotation import is repositioned in the shopping model.

Modified Files

- `lib/data/repositories/products/products_repository.dart`
  - Renamed `_databaseService` to `_dbService` and updated constructor parameter.
  - Switched to `insert<ProductDto>` and `fetchById`/`fetchAll`/`update`/`delete` methods on `_dbService`.
- `lib/data/repositories/shopping/shopping_repository.dart`
  - Renamed `_database` to `_dbService` and updated constructor.
  - Standardized initialization to await `fetchAll` and handle success/failure cases.
  - Swapped to `insert<ShoppingDto>` and `fetchById`/`fetchAll`/`update`/`delete` on `_dbService`.
- `lib/data/services/database/database_service.dart`
  - Added `fetchByFilter`, `updateWhere`, and `deleteWhere` methods for filtered operations.
  - Adjusted `fetchAll` signature to accept nullable `limit` and `offset`.
  - Executed `SqlTables.items` creation during initialization.
- `lib/data/services/database/tables/sql_tables.dart`
  - Extended `Tables` constants with `items`.
  - Expanded `SqlTables` to define `items` table SQL (with composite primary key).
  - Replaced `productShoppingIDIndex` with `productNameIndex` for product name lookups.
  - Updated `ShoppingColumns` and `ProductColumns` to reflect renamed fields.
- `lib/domain/models/shopping/shopping_model.dart`
  - Moved `freezed_annotation` import above DTO import for consistency.

New Files

- `lib/data/repositories/items/i_items_repository.dart` Defines the `IItemsRepository` interface with CRUD and initialization methods for `ItemModel`.
- `lib/data/repositories/items/items_repository.dart` Implements `IItemsRepository`, managing an in-memory map of `ItemModel` instances and persisting via the enhanced `DatabaseService`.

Conclusion

With item support fully integrated and database operations unified, the data layer is now more robust and consistent. The system compiles and item management functions are ready for testing.

2025/06/26 home_view-02 - by rudsonalves

Refactor repository structure to use DTOs, Freezed models, and absolute imports; enhance UI components and update linting

This commit restructures the data layer by introducing `*Dto` classes and migrating existing models to Freezed, streamlines repository interfaces and implementations, and standardizes import paths to absolute references. It also adds UI helpers (button signatures and bottom sheet dialogs), updates lint rules, and refines theme dimensions. The `.gitignore` and analysis options are adjusted, and a Draw.io class diagram is added under `docs/`.

Modified Files

- `.gitignore`
  - Added `*.bkp` pattern to exclude backup files.
- `analysis_options.yaml`
  - Ignored `invalid_annotation_target` analyzer error.
- `lib/config/dependencies.dart`
  - Switched to absolute imports for repository and service files.
- `lib/data/repositories/products/i_products_repository.dart`
  - Updated method signatures to accept `ProductDto` and removed shoppingId parameter.
- `lib/data/repositories/products/products_repository.dart`
  - Introduced `_isInitialized` flag, adapted `insert`, `fetchAll`, `update`, and `delete` to use DTO-to-model mapping and Freezed `fromJson`.
- `lib/data/repositories/shopping/i_shopping_repository.dart`
  - Changed `insert` to accept `ShoppingDto`.
- `lib/data/repositories/shopping/shopping_repository.dart`
  - Refactored `insert`, `fetchAll`, and `update` to use DTOs and Freezed mapping; adjusted default fetch limit.
- `lib/main.dart`, `lib/main_app.dart`, `lib/routing/router.dart`

  - Converted all package imports to absolute (`/…`) paths.

- **`lib/ui/core/themes/dimens.dart`**
  - Increased `spacingVertical` from `6.0` to `12.0`.
- **`lib/ui/core/ui/form_fields/basic_form_field.dart`**
  - Extracted `border` definition into a variable and reused in `InputDecoration`.
- **`lib/ui/view/home/home_view.dart`**
  - Imported new dialog and form-field widgets; added `_newShopping` method to show a bottom sheet for creating purchases.

New Files

- **`docs/Diagrama_de_Classes.drawio`** Draw.io class diagram illustrating the domain entities, DTOs, repositories, services, and view models.
- **`lib/domain/dto/product/product_dto.dart`**, **`.freezed.dart`**, **`.g.dart`** Defines `ProductDto` with factory constructors for creation and JSON serialization.
- **`lib/domain/dto/shopping/shopping_dto.dart`**, **`.freezed.dart`**, **`.g.dart`** Defines `ShoppingDto` analogously for shopping records.
- **`lib/domain/enums/enums.dart`** Introduces `ShoppingType` enum with localized labels.
- **`lib/domain/models/item/item_model.dart`**, **`.freezed.dart`**, **`.g.dart`** New `ItemModel` Freezed class with default quantity.
- **`lib/domain/models/product/product_model.dart`**, **`.freezed.dart`**, **`.g.dart`** Freezed-based `ProductModel` with `fromDto` and `fromJson` factories.
- **`lib/domain/models/shopping/shopping_model.dart`**, **`.freezed.dart`**, **`.g.dart`** Freezed-based `ShoppingModel` mirroring the DTO and domain logic.
- **`lib/ui/core/ui/buttons/button_signature.dart`** Generic `ButtonSignature` class to unify button definitions in dialogs.
- **`lib/ui/core/ui/dialogs/bottom_sheet_dialog.dart`** Reusable bottom sheet dialog widget accepting a title, body content, and button signatures.

Assets and Test Data

- **Draw.io diagram** under `docs/Diagrama_de_Classes.drawio`

Conclusion

All layers have been refactored to use DTOs and Freezed models with consistent import paths. New UI components and lint settings are in place, and the system builds and runs as expected.

2025/06/25 home_view-01 - by rudsonalves

Add core theming, dimensions, fonts, and common UI components

This commit introduces a centralized design system and a suite of reusable UI components. It adds responsive dimension and font abstractions, integrates Google Fonts and Material Symbols, and provides commonly used widgets (buttons, dialogs, form fields, controllers, and utilities). The router import order is also cleaned up to align with provider conventions.

Modified Files

- **lib/routing/router.dart**
  - Moved `provider` import up to group with other package imports.
- **pubspec.yaml**
  - Added `material_symbols_icons: ^4.2815.1` and `intl: ^0.20.2` dependencies for icons and date formatting.

New Files

- **lib/ui/core/themes/dimens.dart** Defines `Dimens` abstraction for mobile and desktop paddings, spacing, and border radii.
- **lib/ui/core/themes/fonts.dart** Declares `FontsTheme` and `AppFontsStyle` for responsive text styles using custom font families.
- **lib/ui/core/themes/util.dart** Adds `createTextTheme()` helper to merge Google Fonts into `TextTheme`.
- **lib/ui/core/ui/buttons/big_button.dart** Reusable large button with loading state and icon support.
- **lib/ui/core/ui/buttons/icon_back_button.dart** Standard back button using Material Symbols.
- **lib/ui/core/ui/dialogs/app_snack_bar.dart** Bottom snack bar helpers for success and error messages.
- **lib/ui/core/ui/dialogs/botton_sheet_message.dart.dart** Modal bottom sheet component with customizable content and actions.
- **lib/ui/core/ui/dialogs/simple_dialog.dart** Alert dialog wrapper with rich text support.
- **lib/ui/core/ui/dismissibles/dismissible_card.dart** Dismissible card widget for edit and delete actions.
- **lib/ui/core/ui/dismissibles/dismissible_container.dart** Styled background container for dismissible widgets.
- **lib/ui/core/ui/editing_controllers/currency_editing_controller.dart** Text controller that masks and parses currency input.
- **lib/ui/core/ui/editing_controllers/masked_editing_controller.dart** Generic mask controller for text fields.
- **lib/ui/core/ui/form_fields/basic_form_field.dart** Styled `TextFormField` wrapper with label, icons, and validation.

- **lib/ui/core/ui/form_fields/date_form_field.dart** Date picker form field with formatted input.
- **lib/ui/core/ui/form_fields/enum_form_field.dart** `FormField` implementation for selecting `Enum` values via toggle buttons.
- **lib/ui/core/ui/form_fields/widgets/toggle_buttons_text.dart** Helper widget for labeled toggle buttons with selection indicator.
- **lib/ui/core/ui/texts/parse_rich_text.dart** Parses markdown-like `*` and `**` syntax in strings and prepends icons for list items.
- **lib/utils/extensions/date_time_extensions.dart** Date formatting and mapping extension methods.
- **lib/utils/extensions/string_extentions.dart** String helpers for digit extraction and date validation.
- **lib/utils/validates/generic_validations.dart** Common validation rules for names, numbers, dates, and phone numbers.

Conclusion

All new theming layers and UI building blocks are in place, providing a consistent and flexible foundation for the app's interface.

2025/06/25 home_view - by rudsonalves

Update Android NDK, integrate shopping & products modules, and apply theming

This commit pins the Android NDK version, registers the shopping repository and adds the products repository interface and stub implementation. Dependency injection is updated to provide the new repositories. The app's theme system is introduced via `MaterialTheme` and a Google Fonts utility, and the main application and router are wired to use the shopping feature. Finally, build dependencies are aligned in `pubspec.yaml`.

Modified Files

- **android/app/build.gradle.kts**
  - Hardcoded `ndkVersion` to `"27.0.12077973"` instead of using `flutter.ndkVersion`.
- **lib/config/dependencies.dart**
  - Imported `IShoppingRepository` and `ShoppingRepository`.
  - Added provider for `IShoppingRepository` with `ShoppingRepository(database)`.
- **lib/data/repositories/shopping/shopping_repository.dart**
  - Changed `shoppingList` getter to return an unmodifiable list.
  - Removed redundant duplicate initialization check in `insert()`.

  - Adjusted default `limit` in `fetchAll()` from `9999` and passed it correctly to the database call.

- **lib/data/services/database/database_service.dart**
  - In `initialize()`, executed `SqlTables.products` and created indexes for `productShoppingIDIndex` and `productBarCodeIndex`.

- **lib/data/services/database/tables/sql_tables.dart**
  - Extended SQL schema: added `shopping_id` and `description` columns to `products` table.
  - Added `productShoppingIDIndex` and `productBarCodeIndex` definitions.

- **lib/main_app.dart**
  - Imported theme and util modules.
  - Initialized a `brightness` field and selected light/dark theme via `MaterialTheme`.

- **lib/routing/router.dart**
  - Injected `IShoppingRepository` into `HomeViewModel` via `context.read`.

- **lib/ui/view/home/home_view.dart**
  - Added elevation to the AppBar for visual depth.

- **lib/ui/view/home/home_view_model.dart**
  - Updated `HomeViewModel` to accept `IShoppingRepository`, initialize with a `Command0`, and expose `shopping` getter.

- **pubspec.yaml**
  - Added `google_fonts: ^6.2.1` dependency.

New Files

- **lib/data/repositories/products/i_products_repository.dart** Defines the `IProductsRepository` interface for product CRUD operations scoped to a shopping list.
- **lib/data/repositories/products/products_repository.dart** Provides a stub implementation of `ProductsRepository` with in-memory caching, `initialize`, `insert`, `fetch`, and `fetchAll`; `update` and `delete` methods marked `TODO`.
- **lib/ui/core/themes/theme.dart** Introduces `MaterialTheme` class with light and dark color schemes (multiple contrast levels) using Material 3 `ColorScheme`.
- **lib/ui/core/themes/util.dart** Adds `createTextTheme()` helper to combine Google Fonts for body and display text styles.

Conclusion

All updates are implemented successfully and the new shopping, products, and theming modules are fully wired and ready for further development.

2025/06/24 shopping_repository - by rudsonalves

Add Makefile, shopping repository, and enhance database service

This commit introduces a Makefile for streamlined development commands, implements a new shopping feature with repository interfaces and implementations, and augments the `DatabaseService` to support ordering, pagination, and extended SQL table definitions. These changes lay the groundwork for CRUD operations on a shopping list.

Modified Files

- **lib/data/services/database/database_service.dart**
  - Added optional `orderBy`, `limit`, and `offset` parameters to query methods.
  - Defaulted `sortBy` to `'created_at ASC'` when no ordering is provided.
  - Passed `orderBy`, `limit`, and `offset` through to `_db!.query`.
- **lib/data/services/database/tables/sql_tables.dart**
  - Renamed standalone `SqlTables` to include a `shopping` table constant.
  - Added `Tables.shopping` and `Tables.products` to the table definitions.

New Files

- **Makefile** Defines common development tasks:
  - `diff` to stage and view pending changes.
  - `push` and `push_branch` to enforce branch protection on `main`/`master`.
  - `rebuild` for Flutter clean and fetch.
  - `test_coverage` and `test_serial` for running tests with coverage.
  - `update_splash` and `update_launcher_icons` to regenerate assets via CLI.
- **lib/data/repositories/shopping/i_shopping_repository.dart** Declares the `IShoppingRepository` interface for shopping list CRUD operations and initialization.
- **lib/data/repositories/shopping/shopping_repository.dart** Implements `ShoppingRepository` with in-memory caching, initialize logic,

and `insert`, `fetch`, `fetchAll`, `update`, and `delete` methods using `DatabaseService`.

Conclusion

All tasks are implemented and the shopping feature is fully integrated and ready for use.

AI-Ready

Sempre que solicitado criar texto de commits usar o prompt a seguir para sua criação:

> Aja como um assistente técnico especializado em desenvolvimento de software. Você receberá um conjunto de mudanças em formato `git diff` de um projeto Flutter. Sua tarefa é gerar um conteúdo para o commit em en_US, com a seguinte estrutura:
>
> - **Título em nível 3 ###:** Comece um título suncito em en_US para o commit (máximo de 80 caracteres).
> - Um parágrafo introdutório em en_US explicando, de forma clara e objetiva, o propósito geral das alterações (em até 4 linhas).
> - **Seção "Modified Files" em nível 3 ###:** Liste todos os arquivos modificados, um por um. Para cada arquivo faça:
>   - gere uma sub-lista com descrição clara das mudanças realizadas nesse arquivo.
> - **Seção "New Files" em nível 3 ###:** (se aplicável) Liste todos os arquivos novos, com uma breve descrição da sua função no projeto.
> - **Seção "Assets and Test Data" em nível 3 ###:** (se aplicável) Liste imagens, arquivos JSON, ou dados de exemplo adicionados.
> - **Seção "Conclusions" em nível 3 ###:** Finalize com uma frase curta indicando que as alterações estão completas e o sistema está funcional.
>
> Regras: - Ignore arquivos auto-gerados (`*.g.dart`, backups, pubspec.lock, etc). - Escreva todo o conteúdo do commit em en_US, com estilo técnico, objetivo e limpo. - Seja preciso com nomes de arquivos e diretórios. - Não use emoji.