Full Stack Developer - Coding Test Documentation

Purpose

The purpose of this coding test is to create a back-office application that allows users to perform CRUD operations on ingredients and items.

Database Schema

Database

- **PostgreSQL** will be used as the database.

`tm_ingredient`

Column	Type	Description
uuid	UUID	Primary key
name	String	Name of the ingredient
cause_alergy	Boolean	Indicates if it causes allergy
type	Int	Values: 0 (none), 1 (veggie), 2 (vegan)
status	Int	Values: 0 (inactive), 1 (active)
created_at	DateTime	Timestamp when created
updated_at	DateTime	Timestamp when updated
deleted_at	DateTime	Timestamp when deleted

`tm_item`

Column	Type	Description
uuid	UUID	Primary key
name	String	Name of the item
price	Decimal	Price of the item
status	Int	Values: 0 (inactive), 1 (active)
created_at	DateTime	Timestamp when created
updated_at	DateTime	Timestamp when updated
deleted_at	DateTime	Timestamp when deleted

`tm_item_ingredient`

Column	Type	Description
uuid_item	UUID	Foreign key referencing `tm_item`
uuid_ingredient	UUID	Foreign key referencing `tm_ingredient`

tablelink

Functional Requirements

Ingredient Management

1. **[Index]** Display a data table that allows setting pagination to 10, 20, or 50 items per page. Displayed columns: name, cause_alergy, type, status.
2. **[Create]** User can input name, cause_alergy (boolean), type, and status. Validate that name must be unique (excluding soft deleted data).
3. **[Update]** User can update all fields but must ensure the name remains unique (excluding the current ID and soft deleted data).
4. **[Delete]** Use soft delete logic.

Item Management

1. **[Index]** Display a data table that allows setting pagination to 10, 20, or 50 items per page. Displayed columns: name, price, status.
2. **[Create]** User can input name, price, status, and a list of ingredients. Validate that name is unique and all fields are required.
3. **[Update]** User can update all fields but must ensure the name remains unique (excluding the current ID and soft deleted data).
4. **[Delete]** Use soft delete logic.

Item-Ingredient Management

1. **[Delete]** Use hard delete logic.

Frontend (FE)

- Developed using React
- Styled with TailwindCSS
- Follows Clean Architecture principles (Uncle Bob)

Backend (BE)

- Developed using Golang
- Uses Fiber as the HTTP framework
- Implements gRPC for managing relationships between tables
- Follows Clean Architecture principles (Uncle Bob)
- Uses `pgxrows` for querying PostgreSQL

Test Focus (Pressure Points)

- **Clean architecture** implementation in both frontend and backend
- **Backend functionality** correctness and efficiency
- **Frontend neatness** in UI and code structure

Note:
GRPC Library: https://grpc.io/docs/languages/go/quickstart/