APILang

Honoré Vicente Cesário
Talison Fernandes Costa

Programa de Pós-graduação em Tecnologia da Informação

Objetivos

- Gerar classes de modelo de uma API, com anotações JPA;
- Descrever o domínio de uma API;
- Gerar versão inicial das classes de modelo.

Motivação

- Diagrama de classes apresentam limitações;
- Ferramentas adicionais são necessárias;

APILang

- Proporciona a descrição do modelo de dados, agregando informações dos diagramas de classe e relacionamento e especificação de requisitos;

- Aproxima implementação do modelo de uma API da especificação.

Contexto

- Aplicação dentro de ambiente de desenvolvimento;
- Uso permitiria que o engenheiro de requisitos, gerente de projeto ou o próprio especialista de domínio pudesse especificar/descrever ou até mesmo prototipar o modelo de dados;
- Aumento da flexibilidade no processo inicial de desenvolvimento.

Implementação

- Tipos das propriedades:
- **primitivo -** property <name> of type <type name>
- **derivado -** property <name> of model <model name>

Restrições

- **required -** campo deve ser preenchido;
- **unique -** identificador único
- **joinColumn -** define detalhes de uma coluna

Relacionamentos entre dois modelos:

- **oneToOne;**
- **oneToMany;**
- **manyToMany;**
- **manyToOne.**

APILang - Estrutura Geral

model **<model name>** package **<package name>** table **<table name>** schema **<schema name>** primary key **<primary key name>** {

property **<property name>** of type|model **<type name>** with constraint **<constraint>**

}

<type name>
String
Integer
Double
Boolean
List<T>

<constraint>
unique
required
oneToOne
oneToMany
manyToMany
manyToOne
joinColumn

Implementação

- **Código gerado contém todos os métodos getters e setters do modelo;**
- **Criado em tempo de execução no caminho especificado.**

java -cp classes:libs/antlr-4.9.1-complete.jar apilang.EvalVisitor <caminho do programa *.api> <caminho do repositório onde será criado o código>

Gramática

```
grammar APILang;

prog: model+ EOF ;

model: 'model' NAME                          # modelName
 | model 'package' PACKAGE_NAME              # modelPackageName
 | model 'table' NAME                        # modelTableName
 | model 'schema' NAME                       # modelSchemaName
 | model 'primary key' NAME                  # modelPrimaryKeyName
 | model '{' property+ '}'                   # modelProperties
 ;
```

Gramática

```
property: 'property' NAME                          # propertyDef
 | property 'of type' TYPE                        # primitiveTypeDef
 | property 'of model' NAME                       # definedModelDef
 | property 'with constraint' constraints+        # constraintsDef
 ;

constraints: 'required' | 'unique' | 'oneToMany' | 'manyToMany' |
'manyToOne' | 'oneToOne' | 'joinColumn' ;

TYPE: 'String' | 'Integer' | 'Boolean' | 'Double' | 'List<'NAME'>' ;

NAME: [a-zA-Z_]+ ;

PACKAGE_NAME: [a-zA-Z.]+ ;

WS : [ \t\r\n]+ -> skip ;
```

Definição do modelo “Categoria”

```
model Category package br.com.fotonica.apilangtest.model table
category schema public primary key id_category {
  property subcategory of type List<Category>
  property isCategory of type Boolean
  property icon of type String
}
```

Código gerado

```
// Arquivo CategoriaModel.java
package br.com.fotonica.apilangtest.model;

import javax.persistence.AttributeOverride;
import javax.persistence.Column;
import javax.persistence.Table;

import java.util.List;


import br.com.fotonica.core.GenericEntity;

@Table(name = "category", schema = "public")
@AttributeOverride(name = "id", column = @Column(name = "id_category"))
public class CategoryModel extends GenericEntity {

    private List<CategoryModel> subcategory;

    private Boolean isCategory;

    private String icon;
```

Código gerado

```
public List<CategoryModel> getSubcategory(){
    return this.subcategory;
}

public void setSubcategory(List<CategoryModel> subcategory){
    this.subcategory = subcategory;
}

public Boolean getIsCategory(){
    return this.isCategory;
}

public void setIsCategory(Boolean isCategory){
    this.isCategory = isCategory;
}

public String getIcon(){
    return this.icon;
}

public void setIcon(String icon){
    this.icon = icon;
}
}
```

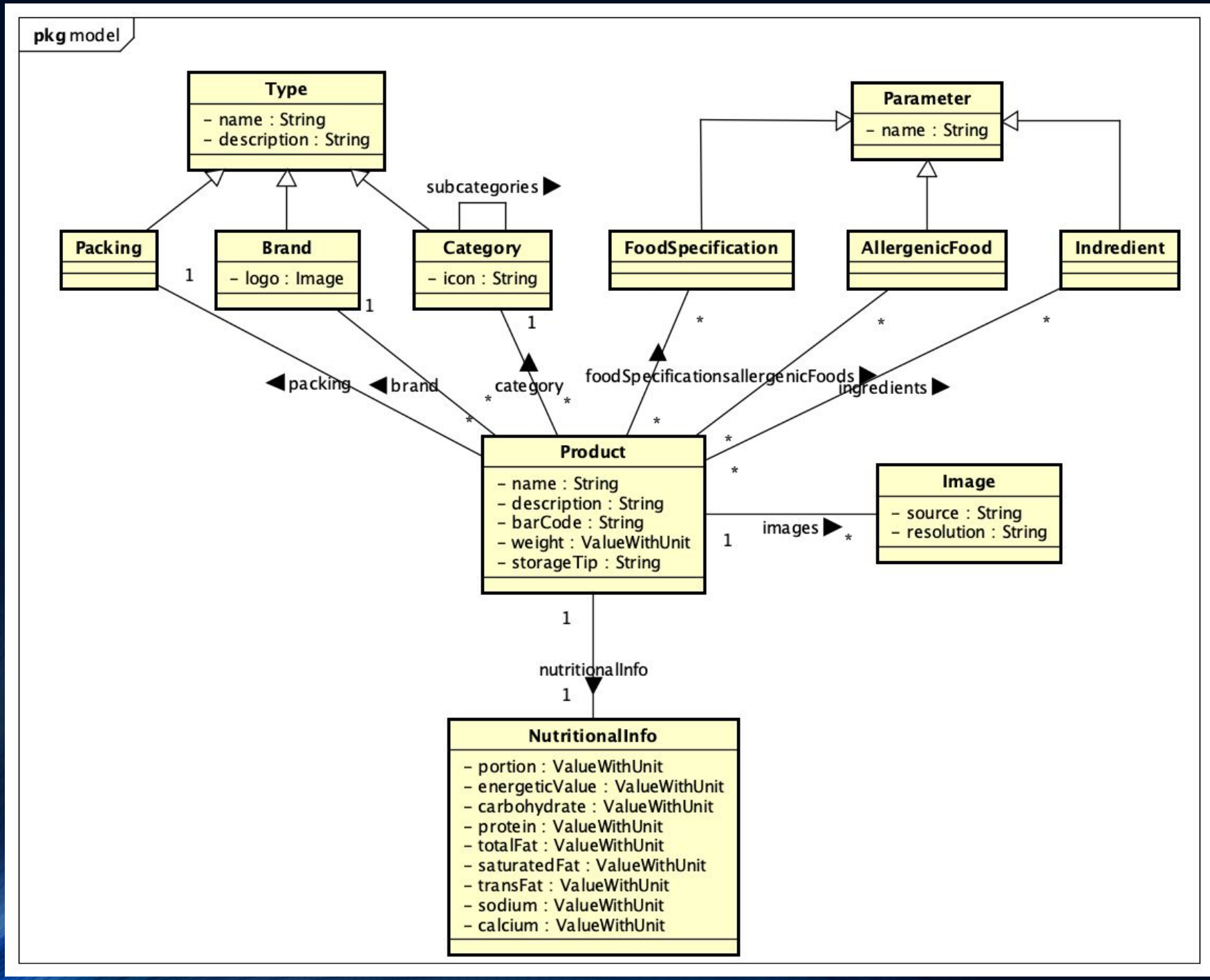
Observações

- Restrições não precisam ser obrigatoriamente especificadas;
- Definição do nome da tabela, schema e chave primária também não são obrigatórias;
- Essas informações podem ser adicionadas posteriormente direto no código gerado

Essas definições visam garantir maior flexibilidade ao arquiteto de software.

Teste de geração de código

O programa escrito especifica as classes do diagrama de domínio de uma API de produtos que é apresentada a seguir.


pkg model
Type
- name : String
- description : String
Parameter
- name : String
subcategories
Packing
Brand
- logo : Image
Category
- icon : String
FoodSpecification
AllergenicFood
Indredient
1
1
1
*
*
*
packing
brand
category
foodSpecifications
allergenicFoods
ingredients
Product
- name : String
- description : String
- barCode : String
- weight : ValueWithUnit
- storageTip : String
Image
- source : String
- resolution : String
images
1
*
1
nutritionalInfo
1
NutritionalInfo
- portion : ValueWithUnit
- energeticValue : ValueWithUnit
- carbohydrate : ValueWithUnit
- protein : ValueWithUnit
- totalFat : ValueWithUnit
- saturatedFat : ValueWithUnit
- transFat : ValueWithUnit
- sodium : ValueWithUnit
- calcium : ValueWithUnit

Especificação do modelo

```
model UnitMeasurement package br.com.fotonica.apilangtest.model table
unit_measurement schema public primary key id_unit_measurement {

  property initials of type String

  property name of type String

}

model ValueWithUnit package br.com.fotonica.apilangtest.model table
value_with_unit schema public primary key id_value_with_unit {

  property value of type Double

  property unitMeasurement of model UnitMeasurement

}

model Category package br.com.fotonica.apilangtest.model table category schema
public primary key id_category {

  property subcategory of type List<Category>

  property isCategory of type Boolean

  property icon of type String

}
```

Especificação do modelo

```
model Brand package br.com.fotonica.apilangtest.model table brand schema public primary key id_brand {
  property name of type String
  property description of type String
}
model Packing package br.com.fotonica.apilangtest.model table packing schema public primary key id_packing {
  property name of type String
  property description of type String
}
model NutritionalInfo package br.com.fotonica.apilangtest.model table nutritional_info schema public primary key id_nutritional_info {

  property portion of model ValueWithUnit with constraint oneToOne joinColumn
```

Especificação do modelo

```
property energeticValue of model ValueWithUnit with constraint oneToOne joinColumn
    property carbohydrate of model ValueWithUnit with constraint oneToOne joinColumn
  property protein of model ValueWithUnit with constraint oneToOne joinColumn
  property totalFat of model ValueWithUnit with constraint oneToOne joinColumn
    property saturatedFat of model ValueWithUnit with constraint oneToOne joinColumn
  property transFat of model ValueWithUnit with constraint oneToOne joinColumn
  property sodium of model ValueWithUnit with constraint oneToOne joinColumn
  property calcium of model ValueWithUnit with constraint oneToOne joinColumn
}
model Product package br.com.fotonica.apilangtest.model table product schema public primary key id_product {
```

Especificação do modelo

```
  property name of type String with constraint unique
  property description of type String
  property barCode of type String
  property weight of model ValueWithUnit
  property storageTips of type String
  property category of model Category
  property packing of model Packing
  property nutritionalInfo of model NutritionalInfo
}
```

- O código em APILang está disponível no GitHub: https://github.com/talisonfc/apilang

Arquivos gerados

- BrandModel.java
- CategoryModel.java
- NutritionalInfoModel.java
- PackingModel.java
- ProductModel.java
- UnitMeasurementModel.java
- ValueWithUnitModel.java

Executando o comando: “**`java -cp classes:libs/antlr-4.9.1-complete.jar apilang.EvalVisitor src/resources/api1.api`**”

Funcionalidades APILang

- Geração de código do modelo de dados em Java com anotações JPA;
- Sintaxe próxima da linguagem “natural”;
- Agrega informações dos modelos de dados em UML através dos DC e ER;
- Elimina a necessidade do uso dos diagramas de ER e Classe.

Trabalhos futuros

A APILang deverá suportar a geração de código das classes controle para especificação das interfaces da API e das classes de serviços para interagir com o banco de dados.