INSTITUTO FEDERAL
São Paulo
Câmpus Campinas

Tecnólogo em Análise e Desenvolvimento de Sistemas
Projeto Interdisciplinar - Arquitetura de Software
Profa. Bianca Pedrosa
4° período – 2021/2

PROJETO INTERDISCIPLINAR: DietaJá

Aluno	Código
Caio Augusto Barretta	CP300354X
Gabriele Leonel	CP3003515
Mízia Sousa Almeida Lima	CP3008118

1. Repositório
https://github.com/caiobarretta/DietaJa

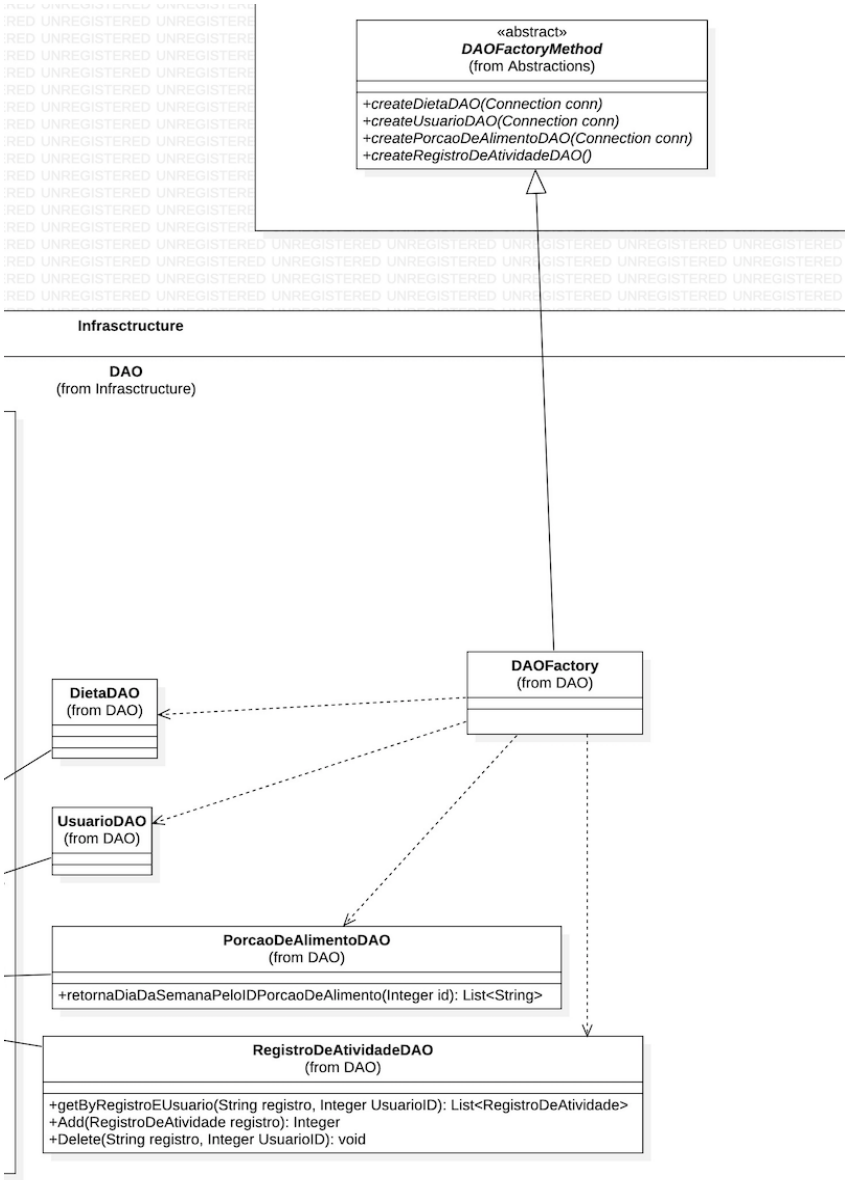
2. Sobre
A aplicação DietaJá é uma app Desktop criada para facilitar a interação entre profissionais da Nutrição e seus pacientes, onde é possível cadastrar novos pacientes, dietas e refeições, bem como interação do paciente com a dieta e posteriormente utilizar essas funcionalidades para o acompanhamento, tanto do paciente quanto do nutricionista.
Importante: As infos aqui disponibilizadas na versão parcial refletem a maior partes das mudanças realizadas no projeto até aqui, outras funcionalidades poderão ser implementadas assim como o desenvolvimento do projeto ainda continua.

3. Padrões Arquiteturais do Projeto
Foram utilizados no projeto até o momento os seguintes padrões: Factory, Strategy, TemplateMethod, ChainOfResponsability.
Além do que são respeitados os princípios de SOLID bem como utilizamos o padrão de arquitetura MVC como design estratégico para fins de abstração foi utilizado o DDD (Domain-Driven Design) que é um modelo de desenvolvimento orientado ao domínio, com fim de abstrair a complexidade do negócio, garantindo que o que está sendo descrito é o que está sendo implementado, neste modelo nós isolamos a arquitetura em camadas, representamos pelo modelo os artefatos de software de maneira bem definida (conhecendo as entities, services, factories, repositories) e gerenciando o ciclo de vida dos objetos do domínio.
Ainda foram implementados alguns testes unitários que serão revisados além de implementados também nos diagramas.
As operações CRUD foram implementadas através de DAO e o Repository para a camada de persistência.

3.1 Factory Method

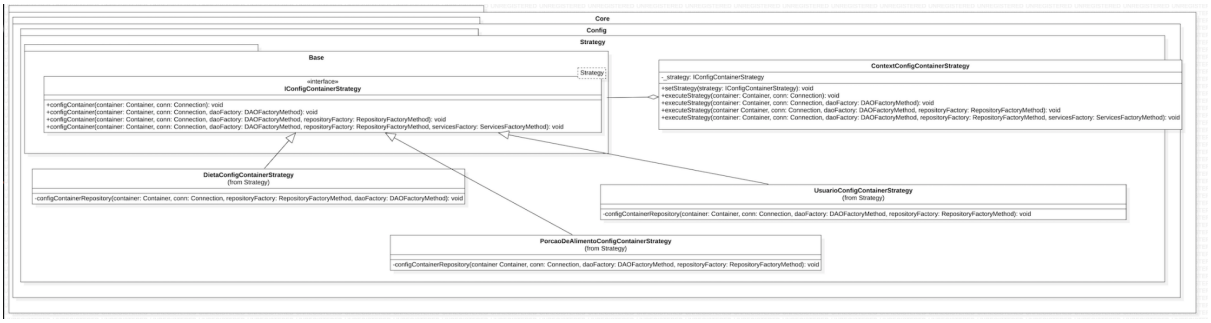
O Factory é um padrão de projeto criacional que visa fornecer uma interface para criar objetos em uma superclasse, e permite também que as subclasses que herdam dela possam alterar o tipo de objetos que estão sendo criados.

No nosso caso o Factory Method do DAO que é uma classe abstracta provê os métodos que serão implementados pela classe concreta de DAO -> DAOFactory que por sua vez através da herança repassa às demais classes que irão utilizar os métodos da classe principal e implementar quaisquer outros que sejam necessários, assim há uma economia de recursos do sistema uma vez que não é necessário implementar concretamente os objetos, nós estamos reutilizando objetos já existentes personalizando conforme a necessidade, há também a implícita aplicação do Princípio de Responsabilidade Única e Open/Closed.



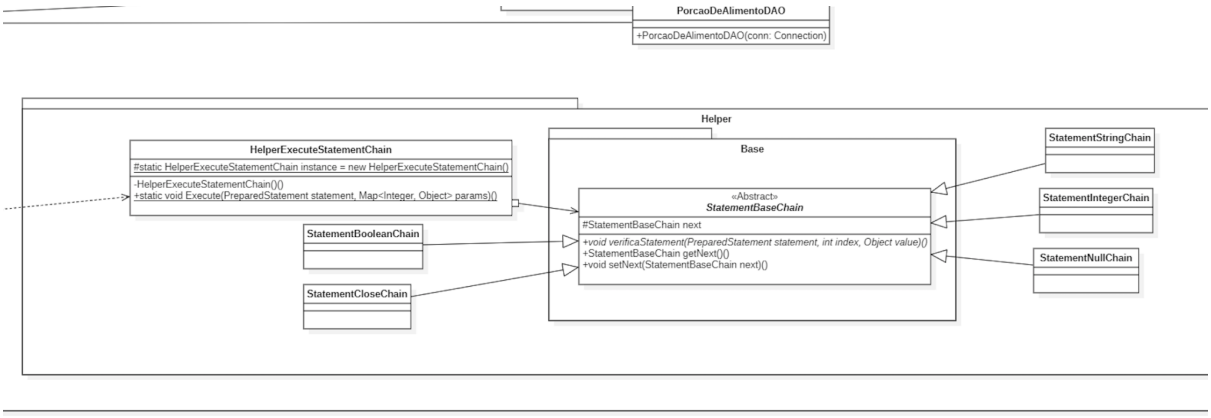
3.2 Strategy

Strategy por sua vez é um padrão comportamental que permite definir uma estratégia para uma determinada família de algoritmos, tornando-os intercambiáveis, em outras palavras nós temos uma classe principal que é a Estratégia e que faz algo específico de diversas maneiras e extraímos essa estratégia para classes separadas. Nós podemos utilizá-la quando temos muitas classes parecidas e que só diferem em algum comportamento, como é o caso do contexto de criação dos containers no nosso código. Nós possuímos a classe **ContextConfigContainerStrategy** que possui as estratégias de start e conexão com o banco e possuímos a Interface **IConfigContainerStrategy** que por sua vez dará as demais classes as estratégias para configurar os Repositorys, como é o caso das demais classes ali herdando: DietaConfigContainerStrategy, PorcaoDeAlimentoConfigContainerStrategy, UsuarioConfig Container Strategy.



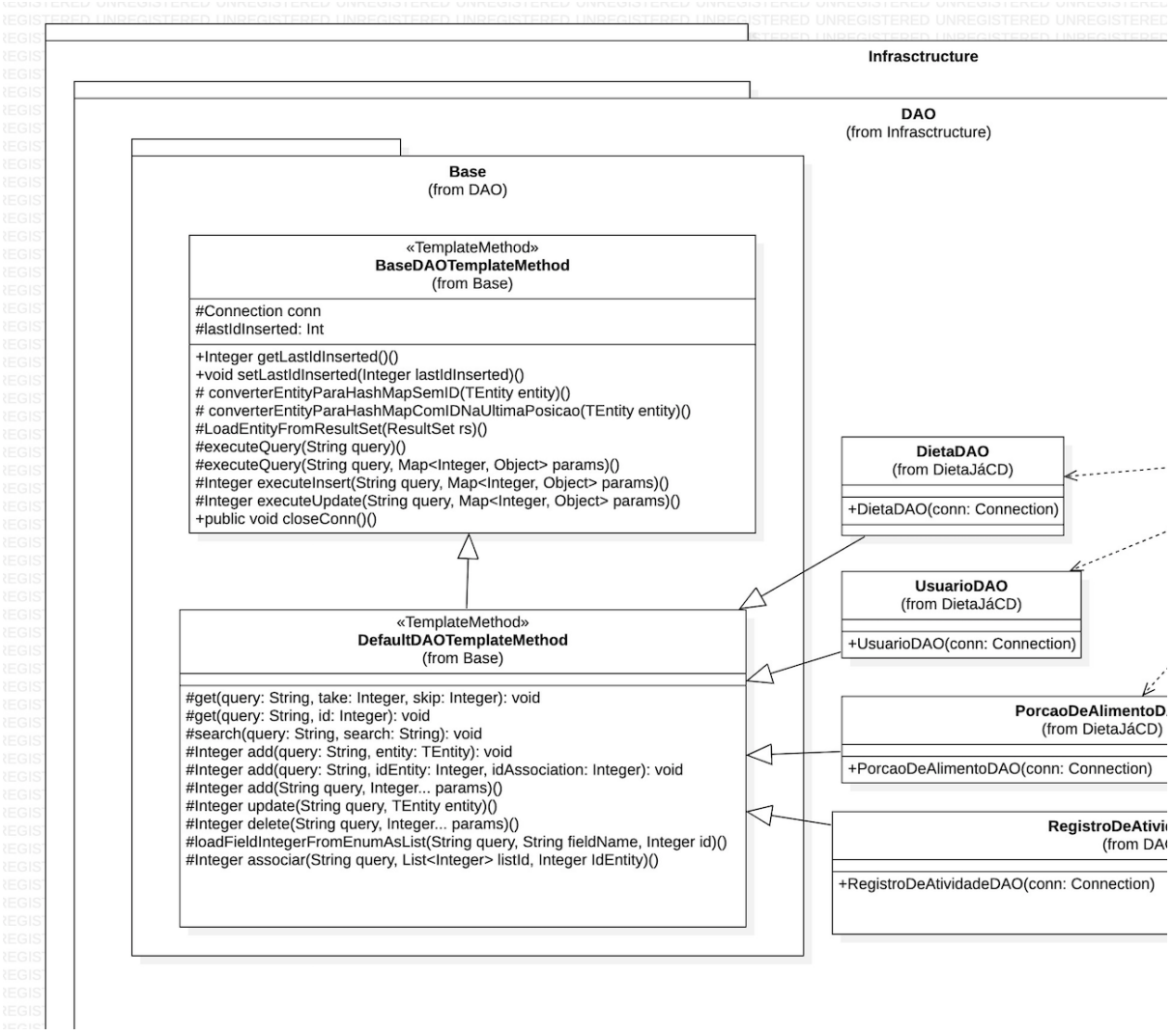
3.3 Chain Of Responsibility

Cadeia de responsabilidades é um padrão de projeto comportamental e ela permite que dependendo da responsabilidade dentro da classe ela passe para a classe que será responsável por executar aquele método. É feita uma checagem. No nosso caso possuímos a **HelperExecuteStatementChain** que possui o método Execute para verificar se o objeto que estamos passando é de algum tipo específico (boolean, integer, null..) e de acordo com esse tipo existe uma exceção que será executada no fim do cadeia conforme, que é o caso das demais classes dessa cadeia (StatementIntegerChain, StatementNullChain..).



3.4 Template Method

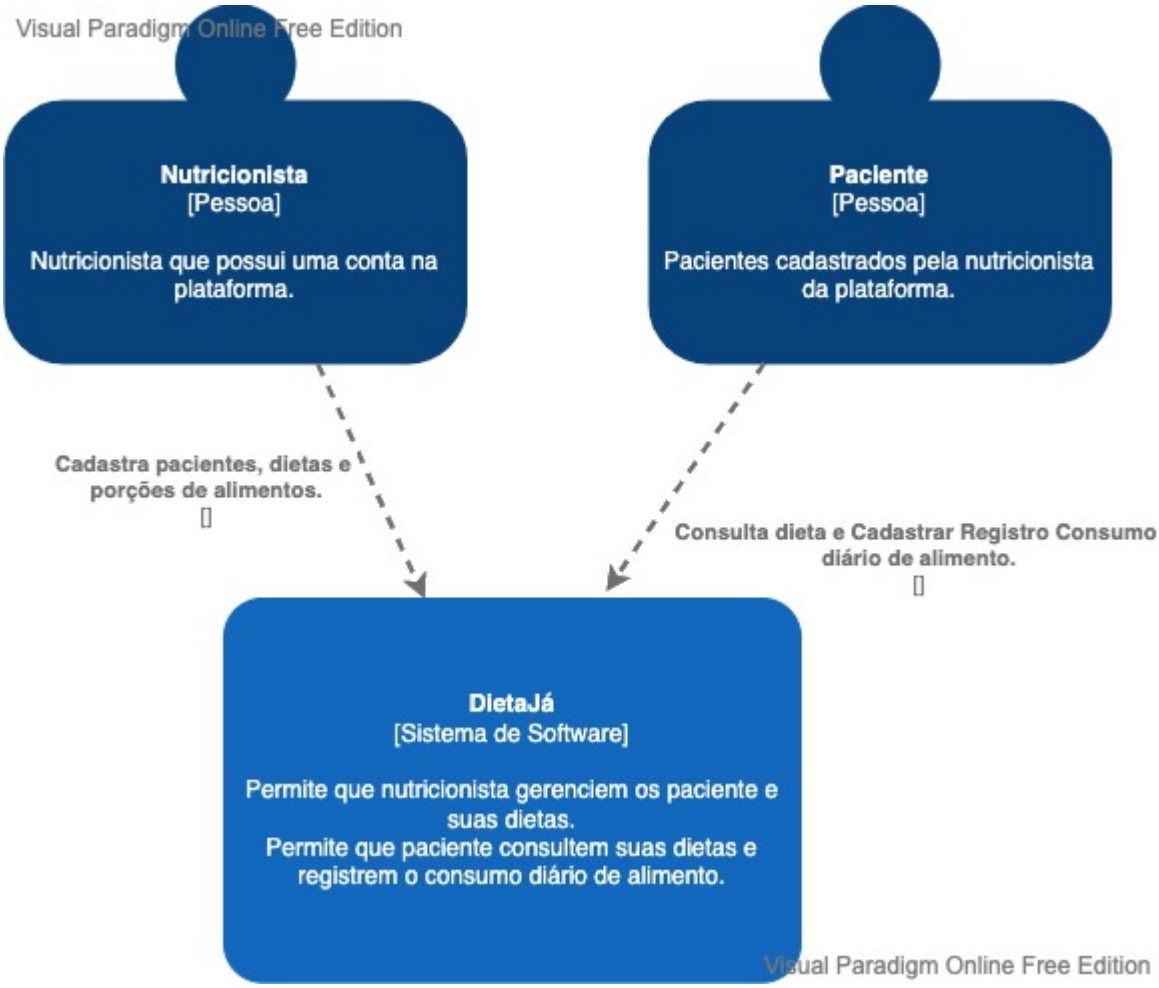
O template method nada mais é que um esqueleto de um algoritmo que vai servir como base para as subclasses que herdam dele sobrescreverem métodos específicos sem modificar sua estrutura, ele vai ser o método padrão e assim todas as classes que herdam delas implementarão todas as etapas abstratas e sobreescrever algumas das opcionais. No nosso caso possuímos o BaseDAOTemplateMethod e o DefaultTemplateMethod, onde nossas classes principais implementam esse template mais seus métodos que sejam necessários, é basicamente a construção de uma casa onde as etapas para construí-la estão ali disponíveis e poderão ser alteradas conforme a necessidade, assim evitamos duplicidade de código.



4. Diagramas

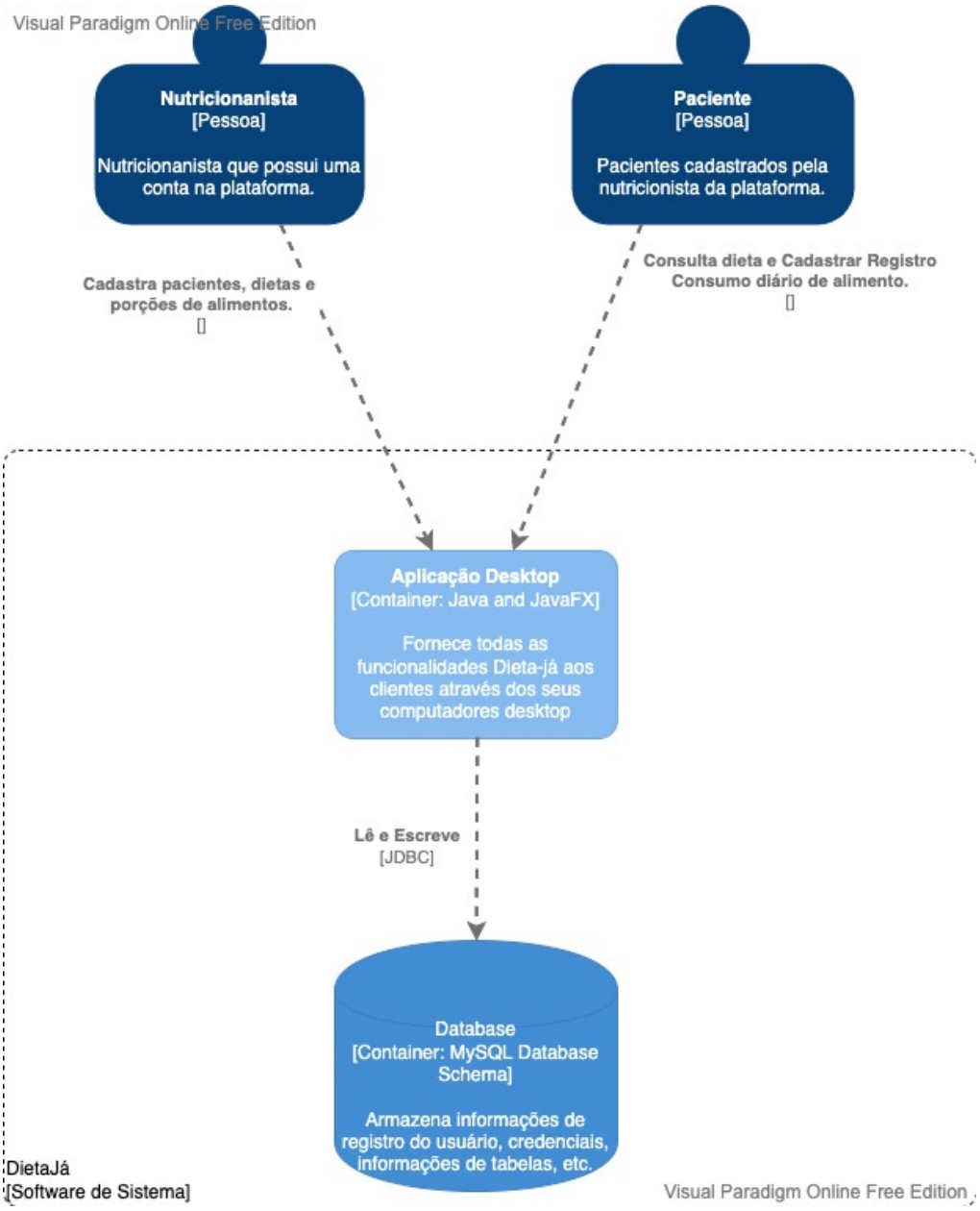
4.1 Diagrama de Contexto

No referido diagrama conseguimos visualizar de maneira objetiva o funcionamento do sistema principal. Possuímos dois atores principais: Nutricionista e Paciente, esses atores interagem com o sistema através das funcionalidades disponíveis a cada perfil, e o sistema em si permite o gerenciamento destas informacoes através das funcionalidades implementadas.



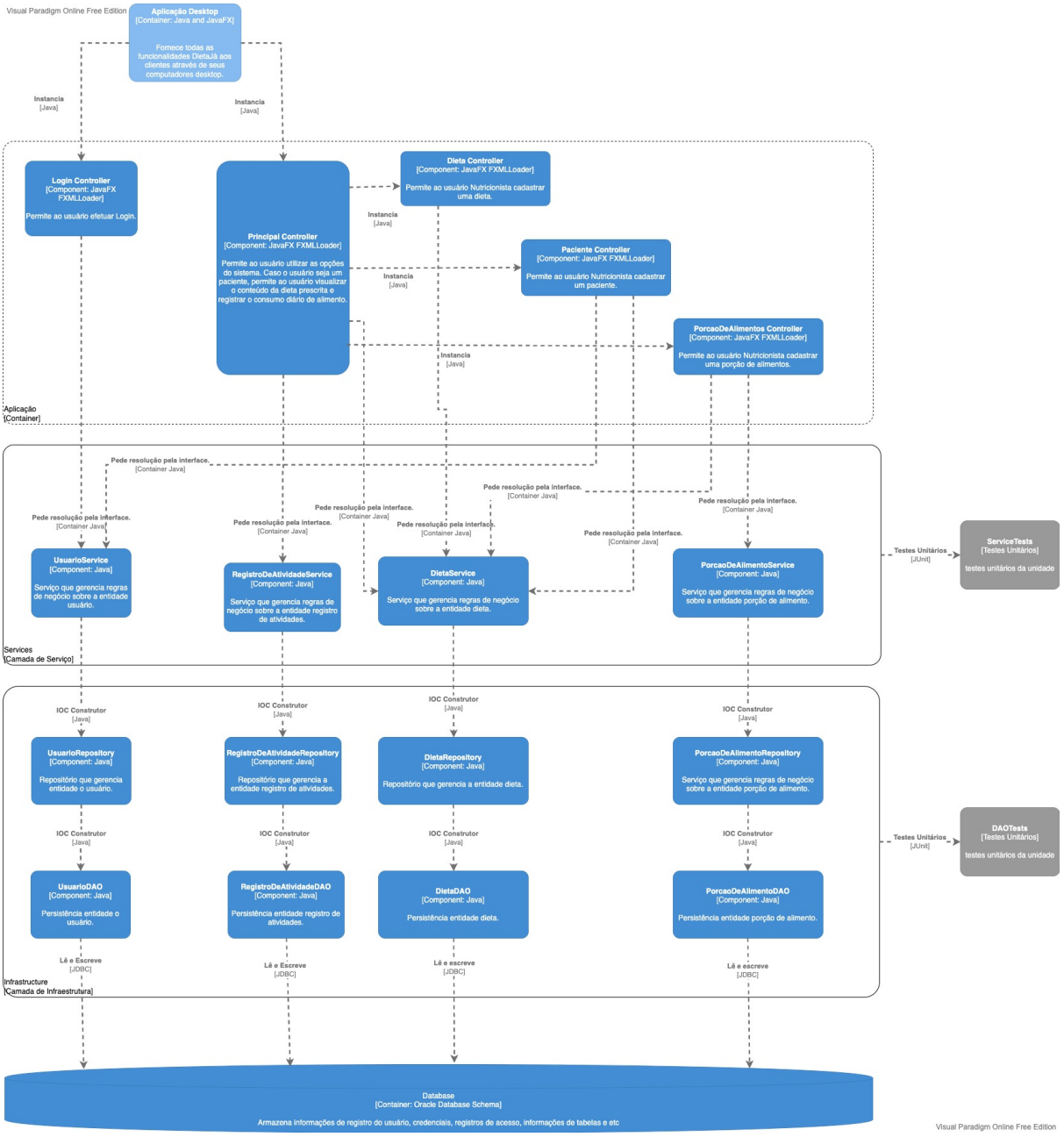
4.2 Diagrama de Container

O diagrama de container mostra os containers existentes na aplicação e possíveis frameworks envolvidos (web, database, APIs e afins). É possível através dessa disposição visual entender as decisões referentes a forma como o código foi construído, suas interações externas e/ou alternativas que hajam da aplicação. No caso do DietaJá, nossos atores Nutricionista e Paciente interagem com a aplicação Desktop e através de CRUD os dados existentes escritos/lidos são armazenados/recuperados a partir de um database, hoje utilizamos o SGBD MySQL.



4.3 Diagrama de Componentes

O Diagrama de componentes amplia os containers e nos dá uma visão micro sobre os componentes existentes em cada container. Para o DietaJá nós temos na representação abaixo e também no repositório. Para tal possuímos a representação da camada de Controllers, Service, Infrastructure - DAO e Repository, e a representação dos testes unitários para as camadas que já possuem os testes.



4.4 Diagrama de Código

O diagrama de código é a representação fiel da implementação do código, é um diagrama UML onde pode conter uma amostra total ou parcial do sistema mostrando os elementos de código pertinentes como interfaces, classes.

Basicamente representa a composição do sistema e sua distribuição de classes, camadas, containers e os detalhes da implementação que refletem o código programado.