INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Cook It

Miguel Achega

Relatório preliminar realizado no âmbito de Projecto e Seminário,
do curso de licenciatura em Engenharia Informática e de Computadores
Semestre de Verão 2020-2021

Orientador : Doutora Matilde Pós-de-Mina Pato

Maio, 2021

Resumo

O projeto Cook It pretende dar resposta a um dos problemas mais frequentes nos nossos lares: O que vamos preparar para o jantar? Trata-se de desenvolver uma aplicação web, recorrendo às tecnologias comuns neste âmbito como java com Spring para desenvolvimento em *back-end*, uma base de dados em PostgreSQL e Vue.js para *front-end*. Um dos desafios vai ser a integração com um outro projeto, Gestão Inteligente de Stocks, que a partir do *stock* que temos na despensa, podemos escolher a nossa refeição.

Índice

Lista de Figuras

Lista de Tabelas

1

Introdução

Quantas horas já passou a pensar no que cozinhar para o almoço ou jantar e no final acabou por fazer a mesma coisa do costume? Acredito que este seja um problema que já aconteceu a muitos de nós, não saber o que cozinhar ou não saber como. O objetivo principal deste projeto é desenvolver uma aplicação web para receitas culinárias para ajudar a combater este problema que tantos de nós temos. Tenho também como objetivo secundário melhorar o meu conhecimento nas tecnologias que irão ser utilizadas e como organizar, documentar e fazer um relatório de um projeto com uma dimensão significativa.

1.1 Especificações do Projeto e Resumo da Solução

Com a Figura 1.1 pretende-se não só apresentar os principais componentes do projeto, bem como demonstrar a relação dos mesmos.

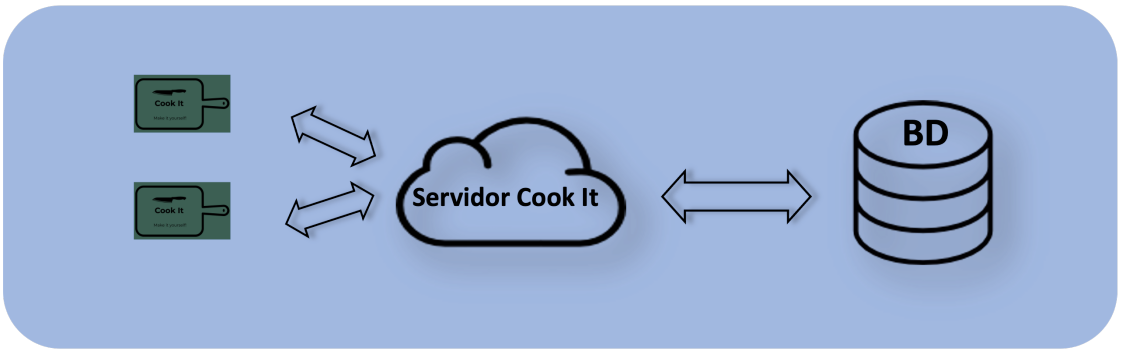
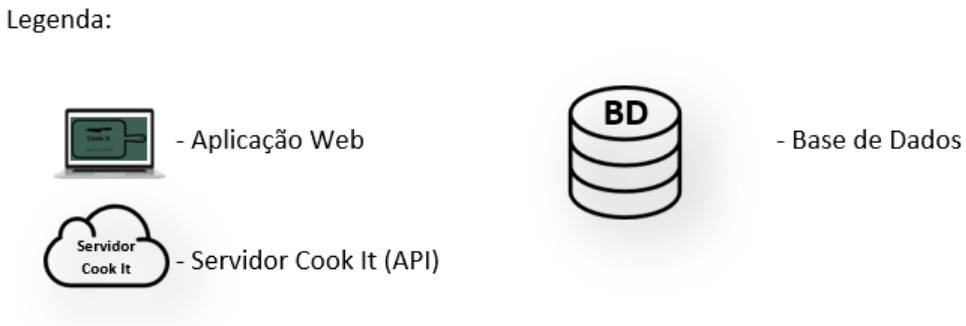


Figura 1.1: Estrutura do Projeto



O projeto é composto por 2 blocos principais que se relacionam. A Figura 1.2 representa esses blocos.

O lado do servidor incluí quatro camadas e expõe uma *API Web*. A camada da Base de Dados (BD) é realizada com o Sistema de Gestão de Base de Dados (SGBD) *PostgreSQL*. A Camada de Acesso a Dados (DAL) é responsável pelas leituras e escritas à BD. Esta camada é produzida com a linguagem de programação *Java*, com *Java Persistent API* (JPA). A Camada da Lógica de Negócio (BLL) é responsável pela gestão dos dados obtidos da BD ou dos *controllers*. A implementação desta camada recorreu à mesma ferramenta que foi usada na DAL. Os *controllers* foram desenvolvidos em *Java* com a *framework* da *Spring*, *Spring Boot*. Do lado do cliente existe um modo de interação, através de uma aplicação web. A aplicação web é disponibilizada para a maioria dos *browsers*, implementada utilizando a linguagem *JavaScript*, com o auxílio da *framework Vue.js*.

Figura 1.2: Arquitectura do Projeto

1.2 Estrutura do Relatório

O relatório está estruturado em 4 capítulos. O capítulo 2 formula o problema, detalhando os requisitos do projeto, e são ainda apresentadas as dificuldades encontradas ao longo do projeto. No capítulo 3 o problema é solucionado, sendo apresentada, com detalhe, a solução implementada. Este capítulo foi dividido pelas várias camadas que compõe o projeto. Na secção 3.1 expõe-se a modelagem dada aos dados. Por conseguinte, a secção 3.2 explica de que forma esses dados foram armazenados, sendo ainda justificadas as decisões tomadas nesta camada. A secção 3.3 fundamenta a seleção das ferramentas utilizadas para a implementação. A lógica de negócio está presente na secção 3.4. E no capítulo 4 retiram-se as conclusões face ao trabalho desenvolvido em relação ao trabalho inicialmente previsto. Para finalizar, propõe-se o trabalho a realizar futuramente, na secção 4.2.

2

Formulação do Problema

Neste capítulo o problema é descrito de forma detalhada na secção 2.1, bem como os requisitos funcionais e não funcionais na secção 2.2. A secção 2.3 apresenta as dificuldades que surgiram no decorrer do projeto.

2.1 Descrição do Problema

Este projeto envolve vários tipos de trabalho como o de desenvolvimento, de avaliação e de resolução de um problema. O projeto consiste no desenvolvimento de uma aplicação web, com uma página simples, intuitiva e *user-friendly*, que permita uma boa experiência de utilizador. Este, terá a possibilidade de criar uma conta e, ao fazer *login* criar uma lista onde poderá guardar as suas receitas preferidas, criar as suas próprias receitas e disponibilizá-las a outros utilizadores, i.e. torná-las públicas na plataforma. Contudo, a plataforma não necessita de uma conta ativa e permite fazer pesquisas de receitas quer a partir do nome, quer a partir da lista de um conjunto de ingredientes. Apenas, os utilizadores registados terão acesso à sua "despensa". Serão consideradas, "todas" as funcionalidades que um utilizador desta plataforma venha a considerar úteis. A internacionalização é também um aspeto importante, pois quero abranger o máximo de utilizadores possíveis, pelo que deverá haver pelo menos duas opções: português e inglês.

2.2 Requisitos Funcionais e Opcionais

Requisitos Funcionais:

- Pesquisa de receitas através do nome;
- *Login* de um utilizador;
- Criação de listas pessoais, para guardar receitas;
- Criação de receitas;
- Pesquisa por outros utilizadores;
- Integração com a aplicação de gestão inteligente de *stocks* de modo a permitir a pesquisa de receitas através dos ingredientes de uma "despensa";

Requisitos Opcionais:

Serão "todos" aqueles que um utilizador desta plataforma venha a considerar úteis, e como tal será necessário fazer um estudo sobre trabalhos relacionados quer a nível nacional como a nível internacional.

2.3 Dificuldades Encontradas

Para a realização deste projeto encontrámos as seguintes dificuldades:

2.3.1 API de receitas só suporta inglês

Um problema encontrado, foi o facto de a API de onde são fornecidas as receitas aos utilizadores só tem suporte para a linguagem inglesa, pelo que não será possível fornecer uma aplicação com ambas as linguagens, português e inglês. No entanto, a aplicação está a ser desenvolvida com suporte a ambas as linguagens, e se no futuro a API das receitas também passar a incluir a língua portuguesa, a aplicação já estará preparada para tal.

3

Solução Proposta

Neste capítulo pretende-se dar ênfase à solução implementada para resolver o problema apresentado no capítulo 1. Este capítulo está divido nas seguintes secções, Modelo de Dados na secção 3.1, Base de Dados na secção 3.2, Acesso a Dados na secção 3.2 e Lógica de Negócio na secção 3.4

3.1 Modelo de Dados

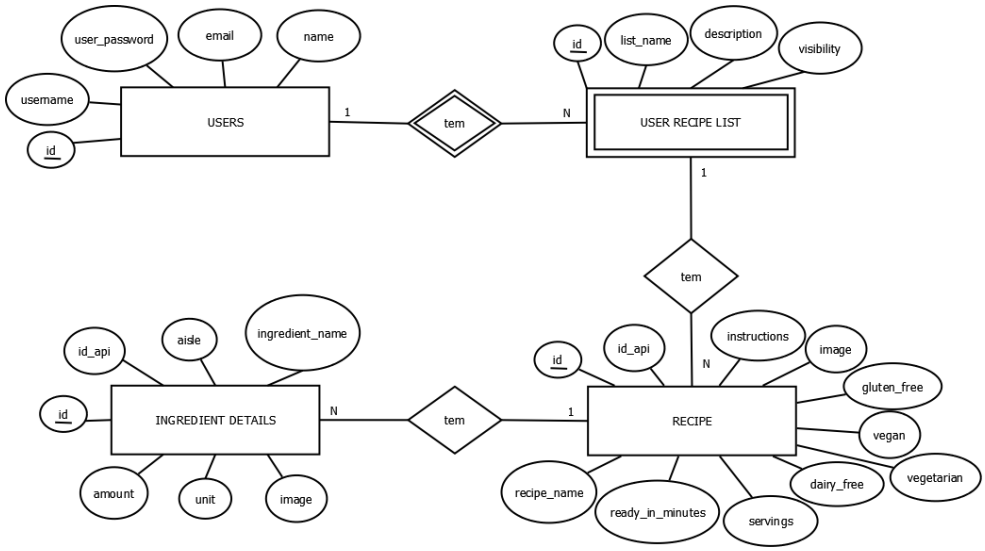


Figura 3.1: Modelo Entidade-Associação

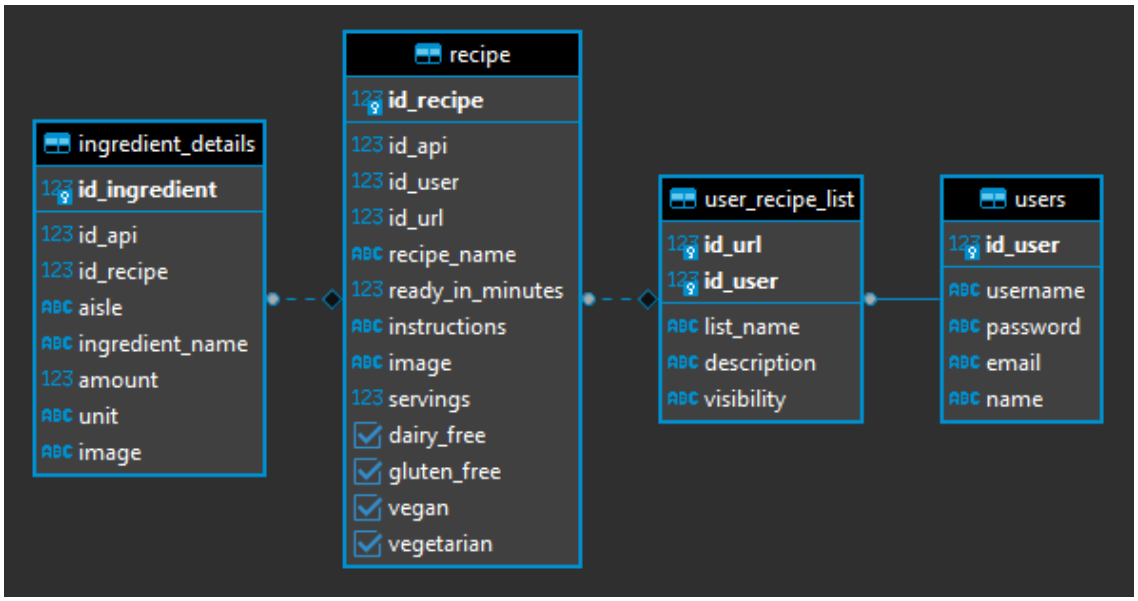


Figura 3.2: Modelo Relacional

Tabela 3.1: Descrição da tabela "Users".

Atributo	**Tipo**	**Restrições Integridade**
id_user	serial	
username	varchar(25)	Único, e até 25 bytes..
password	varchar(25)	Até 25 bytes.
email	varchar(25)	O endereço de email contém "@", e é único. Até 25 bytes.
name	varchar(100)	Até 100 bytes.

Tabela 3.2: Descrição da tabela "Ingredient Details".

Atributo	**Tipo**	**Restrições Integridade**
id_ingredient	serial	
id_api	integer	
id_recipe	integer	id_recipe > 0
aisle	varchar(30)	Não é obrigatório, e até 30 bytes.
ingredient_name	varchar(100)	Até 50 bytes.
amount	double precision	
unit	varchar(20)	Até 20 bytes.
image	text	Não é obrigatório.

Tabela 3.3: Descrição da tabela "Recipe".

Atributo	**Tipo**	**Restrições Integridade**
id_recipe	serial	
id_user	integer	id_user > 0
id_api	integer	
id_url	integer	id_url > 0
recipe_name	varchar(100)	Até 100 bytes.
ready_in_min	smallint	
instructions	text	
image	text	Não é obrigatório.
servings	smallint	
dairy_free	boolean	
gluten_free	boolean	
vegan	boolean	
vegetarian	boolean	

Tabela 3.4: Descrição da tabela "User Recipe List".

Atributo	**Tipo**	**Restrições Integridade**
id_url	serial	
id_user	integer	id_user > 0
list_name	varchar(25)	Até 25 bytes.
description	text	Não é obrigatório.
visibility	varchar(7)	['private', 'public'], 'private' por default.

3.2 Base de Dados

Os dados são armazenados de forma persistente numa Base de Dados (BD). A BD implementada é relacional uma vez que não se preveem alterações durante o uso, ou seja, as tabelas são de certa forma estáticas, não necessitando, portanto do dinamismo oferecido por uma BD documental, por exemplo. A escolha de qual o melhor Sistema de Gestão de Base de Dados (SGBD) assentava em duas possibilidades, *SQL Server* e

PostgreSQL. O primeiro apesar de ser uma ferramenta com a qual estava familiarizado foi excluída visto que um dos requisitos exigidos era ser *open source*, caraterística não presente nesta ferramenta. O *PostgreSQL* por outro lado é *open source* e apresenta as seguintes características:

- O *PostgreSQL* é compatível com as propriedades *Atomicity, Consistency, Isolation, 22Durability (ACID)*, garantindo assim que todos os requisitos sejam atendidos;
- O *PostgreSQL* aborda a concorrência de uma forma eficiente com a sua implementação de *Multiversion Concurrency Control (MVCC)*, que alcança níveis muito altos de concorrência;
- O *PostgreSQL* possui vários recursos dedicados à extensibilidade. É possível adicionar novos tipos, novas funções, novos tipos de índice, etc;

Assim sendo, foi escolhido o Sistema de Gestão de Base de Dados Relacional de Objetos (SGBDRO) *PostgreSQL*.

3.3 Acesso a Dados

Uma vez armazenados os dados de forma persistente é indispensável realizar escritas e leituras sobre os mesmos. Para tal, desenvolveu-se a chamada Camada de Acesso a Dados (DAL). Para implementar esta camada, foi utilizado o *Java Persistent API* (JPA). Tal, permite reduzir a extensa repetição de código envolvido para suportar as operações básicas de *Create, Read, Update e Delete* (CRUD) em todas as entidades. Aqui, o requisito é o acesso aos dados na BD e o suporte para as operações CRUD em quase todas as tabelas. Desta forma criou-se uma *interface Repository* com métodos que garantem não só essas operações, como outras para facilitar a obtenção de dados de determinada maneira. Existe ainda a possibilidade de criar *queries*, definindo métodos nas interfaces JPA. O uso de JPA obriga a representar o esquema/modelo da BD em classes *Java, Plain Old Java Objects* (POJO).

3.3.1 Implementação

No acesso a dados, são utilizados dois padrões de desenho: Padrão *Repository* e Padrão *Unit Of Work*. Esta componente é, salvo exceções, gerada através da JPA. Cada entidade presente na BD é mapeada numa classe em *Java*, que representa o modelo da

mesma. Esta classe tem várias anotações da JPA para referir a Chave-Primária, Chave-estrangeira, relações entre entidades, etc. Em conjunto estas classes *Java* formam o modelo utilizado entre as camadas internas do lado do servidor. Mais à frente serão apresentados outros tipos de objeto usados para representar as entidades recebidas e enviadas para o exterior.

3.4 Lógica de Negócio

É fundamental fazer cumprir as regras, restrições e toda a lógica da gestão dos dados para o correto funcionamento das aplicações. Assim este controlo foi depositado na camada da lógica de negócio (BLL) e também no modelo desenvolvido. Esta decisão permite não só concentrar a gestão dos dados como também controlar numa camada intermédia os dados a obter, atualizar, remover ou inserir, antes de realizar o acesso/escrita dos mesmos.

3.4.1 Implementação

Foram criados serviços para as principais entidades, que dispõem de diversas funcionalidades. É de salientar que um serviço está fortemente ligado a um ou mais repositórios.

4

Conclusões

Neste capítulo apresentam-se as conclusões relativas ao desempenho e trabalho realizado. São efetuadas comparações face ao planeamento inicial previsto e ao que realmente sucedeu, como forma de analisar e apreciar o trabalho realizado.

4.1 Sumário

Nas semanas iniciais foi realizada pesquisa de forma a melhor entender conceitos, dificuldades e potenciais resoluções e/ou abordagens. De seguida definiu-se o problema e como seria solucionado, tendo também sido apresentada a proposta de projeto publicamente. A partir das seguintes datas, começou-se a implementação das várias camadas. Até à data foram desenvolvidas as camadas: Base de Dados, Acesso a Dados, Lógica de Negócio e iniciada a API Web. Também foi desenvolvido o logótipo e cartaz do projeto. De forma geral os requisitos foram cumpridos.

4.2 Trabalho Futuro

Existe ainda trabalho crucial por realizar, nomeadamente a API Web, a aplicação web e a integração com a aplicação de gestão inteligente de stocks, pelo que ainda será necessário bastante esforço e trabalho para realizar as restantes tarefas.