



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

Junho, 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	vii
Lista de Tabelas	ix
1 Introdução	1
1.1 Especificações do Projeto e Resumo da Solução	2
1.2 Estrutura do Relatório	3
2 Formulação do Problema	5
2.1 Descrição do Problema	5
2.2 Requisitos Funcionais e Opcionais	6
2.3 Dificuldades Encontradas	6
2.3.1 API de receitas só suporta inglês	6
2.3.2 API de receitas tem número limitado de pedidos diários	7
2.3.3 Pouca experiência e conhecimento na tecnologia de front-ent	7
3 Solução Proposta	9
3.1 Modelo de Dados	9
3.2 Base de Dados	11
3.3 Controladores	12
3.4 Formato dos Erros	12

3.5	Acesso a Dados	13
3.5.1	Implementação	13
3.6	Lógica de Negócio	13
3.6.1	Implementação	14
3.7	Segurança	14
3.7.1	<i>Cross-Origin Resource Sharing</i>	15
3.7.2	Armazenamento de <i>Passwords</i>	16
3.8	Aplicação Web	16
3.8.1	Internacionalização	16
3.8.2	Segurança	16
3.8.3	<i>Services</i>	16
3.8.4	Componentes	17
3.8.5	<i>Router</i>	17
3.8.6	Armazenamento de Dados	17
4	Conclusões	19
4.1	Sumário	19
4.2	Trabalho Futuro	19

Lista de Figuras

1.1	Estrutura do Projeto	2
1.2	Arquitectura do Projeto	3
3.1	Modelo Entidade-Associação	9
3.2	Modelo Relacional	10
3.3	Funcionamento do JSON Web Token	14
3.4	Configuração Spring Security	15

Lista de Tabelas

3.1	Descrição da tabela “Users”.	10
3.2	Descrição da tabela “Ingredient Details”.	10
3.3	Descrição da tabela “Recipe”.	11
3.4	Descrição da tabela “User Recipe List”.	11



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.

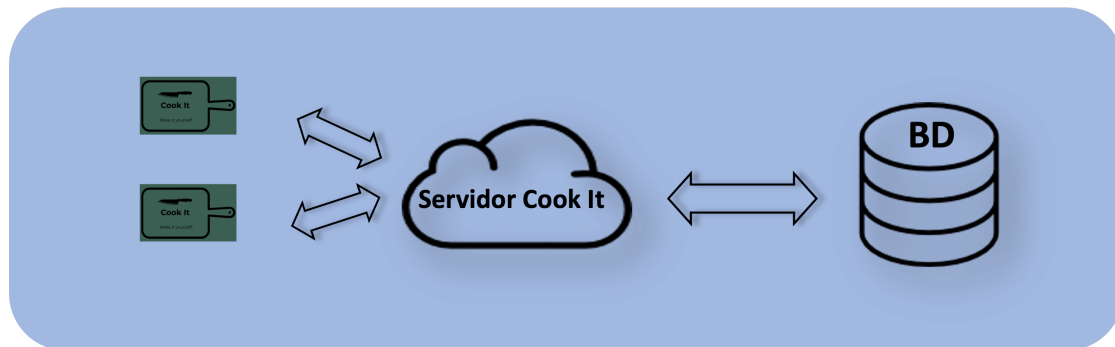
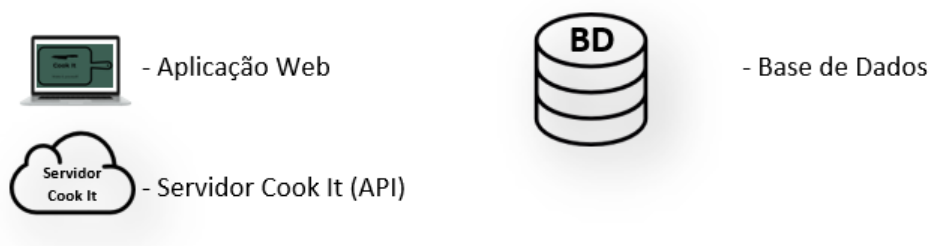


Figura 1.1: Estrutura do Projeto

Legenda:



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

O lado do servidor inclui 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*, , tratando-se de uma *Single Page Application*.

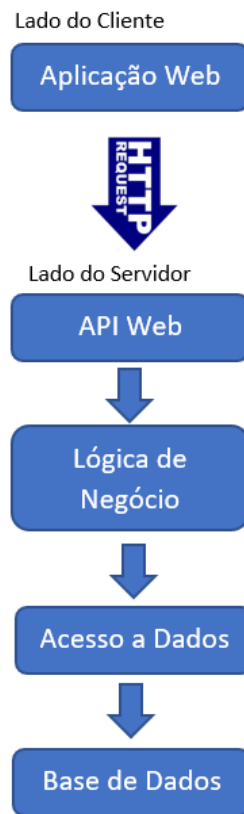


Figura 1.2: Arquitectura do Projeto

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, primeiramente falando sobre a base de dados, de seguida as implementações da API web e por fim a implementação da aplicação cliente.

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.

2.3.2 API de receitas tem número limitado de pedidos diários

Outro problema encontrado ao começar a desenvolver a camada cliente, foi a limitação imposta pela API de receitas no número de pedidos diários que são possíveis efectuar com uma subscrição grátis. Só são possíveis efectuar 50 pedidos diários, pelo que o desenvolvimento tornou-se um pouco mais difícil e demorado.

2.3.3 Pouca experiência e conhecimento na tecnologia de front-end

Um dos problemas com que me deparei quando quis começar a implementar o *front-end* da aplicação foi o pouco conhecimento e experiência que tinha com a tecnologia, *Vue.js*. Então optei por dedicar cerca de semana e meia a duas semanas a aprender e adquirir um pouco de experiência com esta tecnologia, vendo tutorias e cursos online. Agora posso dizer que foi um obstáculo ultrapassado com sucesso e adquiri novos conhecimentos de uma tecnologia muito utilizada nos dias de hoje!

Solução Proposta

Neste capítulo pretende-se dar ênfase à solução implementada para resolver o problema apresentado no capítulo 1. Este capítulo está dividido 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 3.6

3.1 Modelo de Dados

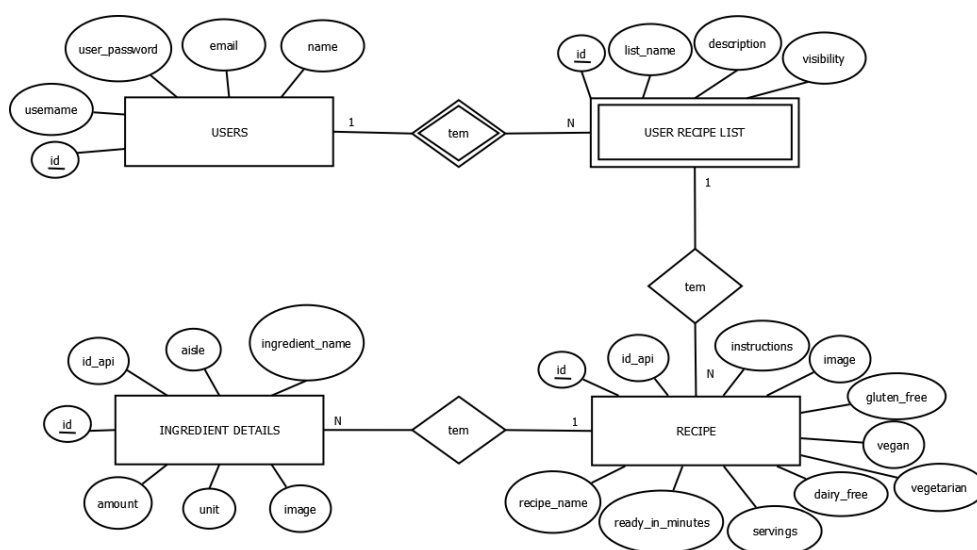


Figura 3.1: Modelo Entidade-Associação

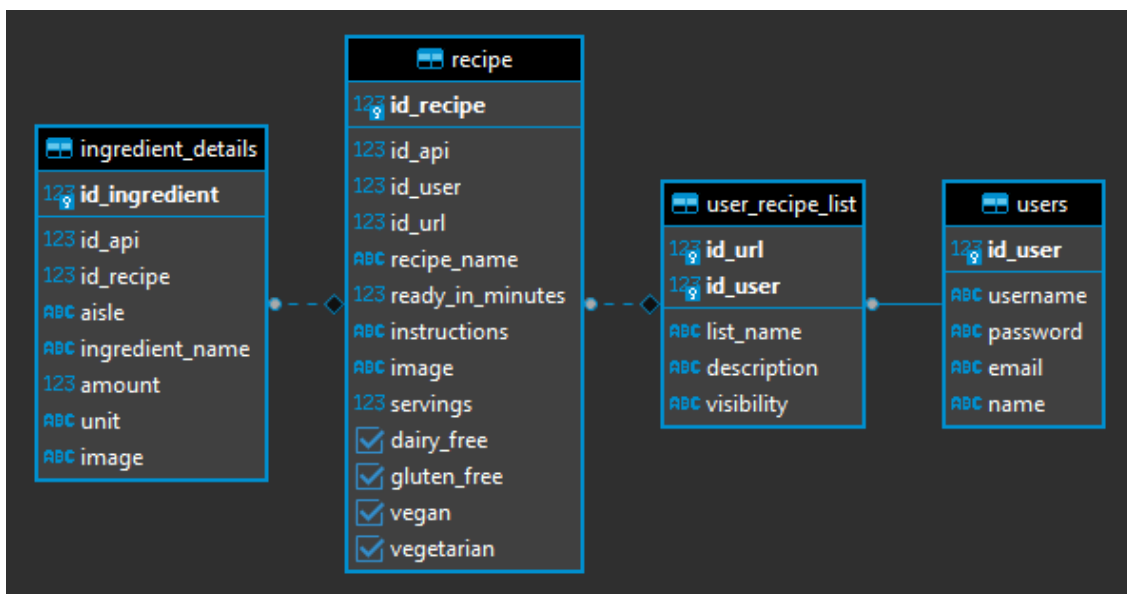


Figura 3.2: Modelo Relacional

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 prevêem 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*, característica 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 Controladores

Os *controllers* identificam os pedidos e direcionam-os para os serviços adequados ao seu processamento, retornando no final a resposta apropriada. Os formatos de resposta utilizam *hypermedia*, e estão a ser utilizados dois tipos: *Siren* que representa respostas em que o *status code HTTP* representa sucesso e *Problem Details* para representar respostas em que o *status code HTTP* representa erro. A escolha do uso de *hypermedia* apoia-se em questões evolutivas da API em termos de hiperligações, ou seja, caso os *endpoints* dos recursos sejam alterados a aplicação cliente não sofre alterações. Cheguei à conclusão que usar *controllers* com *hypermedia* seria mais vantajoso para o projeto, pois as regras de negócio são definidas pela API e estão embebidas nas representações em *hypermedia*.

3.4 Formato dos Erros

De forma a uniformizar os erros expostos pela API, utilizou-se *hypermedia Problem Details*. Esta *hypermedia* permite dar ao utilizador mais informação sobre o erro e como resolve-lo, caso seja um erro de cliente. Todos os pedidos realizados à API em que ocorra um erro, é retornada uma representação no formato *Problem Details*.

3.5 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).

3.5.1 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.

3.6 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.

3.6.1 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.

3.7 Segurança

O mecanismo escolhido para autenticação foi JWT (JSON Web Token), em conjunto com o Spring Security, o que tornou o processo mais simples. Hoje em dia, o JWT é dos mais utilizados para autenticação e troca de informações. Em vez de criar uma Sessão (autenticação baseada em sessão), o servidor codifica os dados em um JSON Web Token e envia-os para o cliente. O cliente salva o JWT e, de seguida, cada solicitação do cliente para rotas ou recursos protegidos deve ser anexada a esse JWT (geralmente no header). O servidor então valida esse JWT e retorna a resposta.

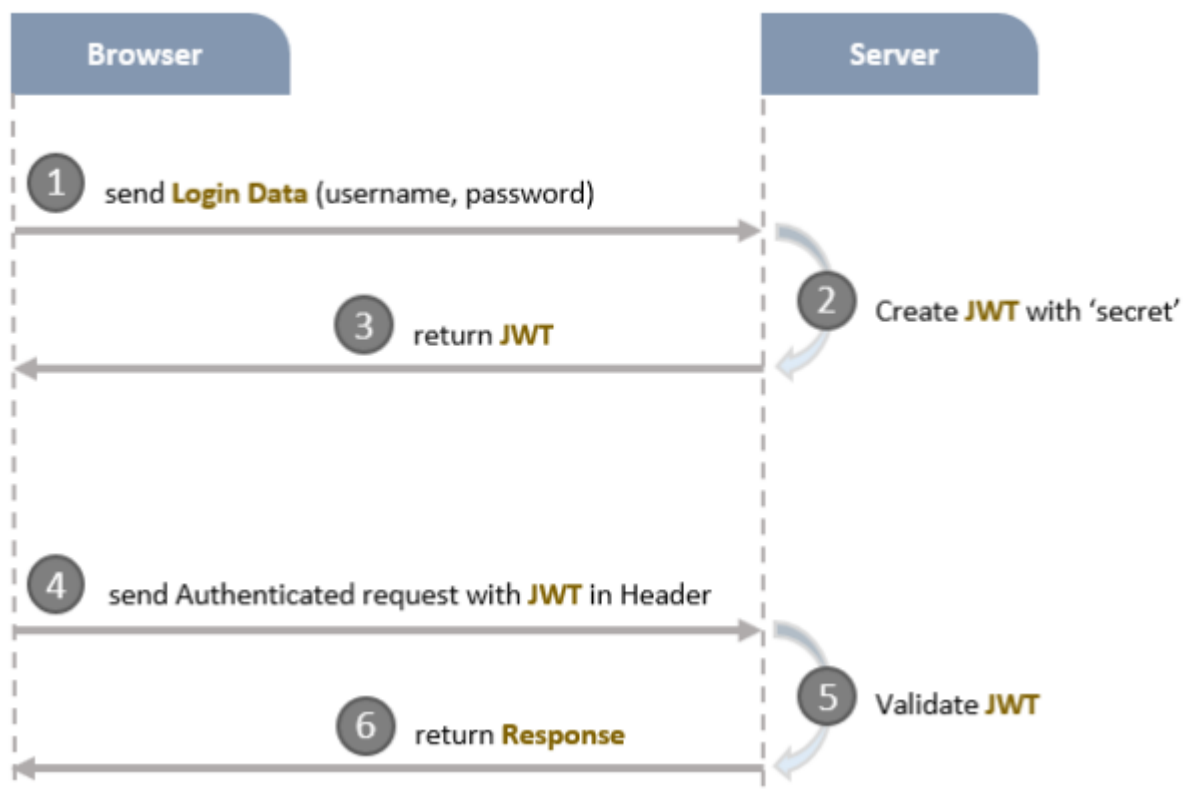


Figura 3.3: Funcionamento do JSON Web Token

Comparando com a autenticação baseada em sessão que precisa de armazenar a sessão no cookie, a grande vantagem do JWT (autenticação baseada em token) é que armazenamos o token no lado do cliente: local storage para o browser (e foi este o utilizado), Keychain para IOS e Shared Preferences para Android. Assim, não é preciso criar outro projeto de back-end que suporte aplicações nativas ou um módulo de autenticação adicional para utilizadores de aplicações nativas.

3.7.1 *Cross-Origin Resource Sharing*

Ao desenvolver uma API que possa ser acessível através de pedidos que tenham um endereço diferente do endereço da API, é preciso configurar a API para responder de forma a que o resultado possa ser visível no *browser*. A especificação *World Wide Web Consortium* (W3C) para *Cross Origin Resource Sharing* (CORS), permite que os resultados se tornem visíveis, evitando assim a política de segurança imposta pelos *browsers*. Assim deu-se permissão para todos os endereços e todos os *headers HTTP*, de forma a que os utilizadores da API possam aceder livremente a esta através do *browser*. Com auxílio do *Spring Security* a configuração é realizada como se mostra na seguinte figura.

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.cors().configurationSource(corsConfigurationSource())
            .and().csrf().disable()
            .exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and()
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
            .authorizeRequests().antMatchers(HttpMethod.POST, "/v1/users").permitAll()
            .antMatchers(HttpMethod.POST, "/v1/users/login").permitAll()
            .anyRequest().authenticated();

        http.addFilterBefore(authenticationJwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);
    }

    private static CorsConfigurationSource corsConfigurationSource() {
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        CorsConfiguration config = new CorsConfiguration();
        config.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "PATCH", "DELETE"));
        config.setAllowedOrigins(Collections.singletonList("*"));
        config.setAllowedHeaders(Collections.singletonList("*"));
        source.registerCorsConfiguration(pattern: "/*", config);
        return source;
    }
}
```

Figura 3.4: Configuração Spring Security

3.7.2 Armazenamento de *Passwords*

É considerado má prática o armazenamento de informação sensível, tal como, as de *passwords* não encriptadas na base de dados, então estas devem ser encriptadas antes de serem armazenadas. Para a encriptação é feito o *hash* da palavra-chave, adicionando um valor aleatório. A este valor aleatório dá-se o nome de *salt*. Com o uso de *salt* previne-se que palavras-chave iguais sejam armazenadas na base de dados com valores iguais. Caso alguém consiga aceder à base de dados não conseguirá descobrir a palavra-passe de um determinado utilizador, isto é garantido encriptando a palavra-chave. Para encriptar as palavras-chave usou-se uma implementação do algoritmo *BCrypt* fornecido pelo *Spring Security*. Na implementação do *Spring Security* o *salt* é gerado internamente, ficando concatenado com a *password* cifrada.

3.8 Aplicação Web

Esta aplicação é disponibilizada para dispositivos *desktop*, através do *browser*. Na sua implementação, teve-se em atenção conceitos como *responsive design*, algo fácil de obter através da utilização de *Vue*, pois tal permite, não só, uma melhor experiência de utilizador, como também, uma interface mais apelativa.

A aplicação web foi pensada como uma aplicação de consulta, onde facilmente os utilizadores podem ter acesso a informação sobre receitas através dos *browsers*: *Google Chrome*, *Microsoft Edge*, *Mozilla Firefox* e *Opera*.

3.8.1 Internacionalização

A aplicação *web* está disponível, apenas, no idioma Inglês, pois este é o unico idioma suportado pela API de receitas.

3.8.2 Segurança

A autenticação é efetuada nos pedidos através do *header HTTP Authorization*. O token é guardado através de *Session Storage*.

3.8.3 Services

Neste módulo estão todos os ficheiros, organizados pelas respectivas entidades, que fazem os respectivos pedidos à API servidora desenvolvida previamente. Os pedidos

são feitos utilizado a biblioteca *axios*.

3.8.4 Componentes

Os componentes são um dos recursos mais poderosos do *Vue.js*. Ajudam a estender os elementos *HTML* básicos para encapsular código reutilizável. Podem ser reutilizados quantas vezes forem necessárias ou usados noutra componente, tornando-o um *child component*.

Na pasta *pages* encontram-se os componentes que representam uma página na aplicação web, sendo cada uma destas 'páginas' mapeada por um router. Na pasta *components* encontram-se os restantes componentes que não são páginas mas são necessários a outros componentes, sendo estes então *child components*. Desta forma foi possível manter o código organizado, fácil de perceber, flexível e mais curto.

3.8.5 Router

Para o tratamento das rotas da aplicação web, foi utilizada a ferramenta *vue-router*. É o router oficial do *Vue.js* e torna a criação de uma *Single Page Application*, como esta, muito mais rápido e fácil. No ficheiro *router.js* encontram-se todas as rotas existentes na aplicação web, mapeadas aos respectivos componentes. Assim quando uma daquelas rotas for inserida o *Vue* sabe exatamente qual o componente que deve mostrar.

3.8.6 Armazenamento de Dados

Para o armazenamento de dados na aplicação cliente foi utilizada a biblioteca *vuex*. O *vuex* dá-nos a capacidade de armazenar e partilhar dados reativos em toda a aplicação sem comprometer o desempenho, testabilidade ou manutenção. Aumenta o sistema de reatividade do *Vue* e cria dados facilmente acessíveis entre componentes independentes. Deste modo, o *vuex* foi a ferramenta escolhida para armazenar os dados na aplicação cliente. Como tal foi criada uma pasta, *store* que contém dois módulos, *recipes* e *users*. Cada um destes módulos é responsável por guardar os respectivos dados. Em *recipes* guardamos as receitas, e as listas de receitas para que as possamos mostrar na aplicação de forma reactiva quando necessário. Em *users* guardamos as informações relativas a um utilizador como por exemplo se este está *logged in* ou detalhes do utilizador, como o seu nome, email, etc...

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. Foram desenvolvidas as camadas: Base de Dados, Controladores, Acesso a Dados, Lógica de Negócio, tratamento de erros e exceções, autenticação, tanto no servidor como no cliente e a API Web. Foi criada uma coleção no Postman com todos os endpoints da API Web, para que esta pudesse ser testada à medida que ia sendo desenvolvida. Também foi iniciado o desenvolvimento da aplicação web e concluído o logótipo e cartaz do projeto.

4.2 Trabalho Futuro

Ainda existem algumas tarefas que precisam de ser realizadas, como terminar a aplicação web e integrar com a aplicação de gestão inteligente de stocks em que iremos

passar a ter uma nova funcionalidade de pesquisa de receitas através dos ingredientes na nossa despensa.