ISABELA MARA DO NASCIMENTO

DESENVOLVIMENTO DE UM DASHBOARD PARA A VISUALIZAÇÃO DE CARDÁPIOS EM UM RESTAURANTE UNIVERSITÁRIO

LAVRAS – MG

2022

ISABELA MARA DO NASCIMENTO

DESENVOLVIMENTO DE UM DASHBOARD PARA A VISUALIZAÇÃO DE CARDÁPIOS EM UM RESTAURANTE UNIVERSITÁRIO

Relatório técnico apresentado à Universidade Federal de Lavras como parte das exigências do curso de Sistemas de Informação, para a obtenção do título de Bacharel

Prof. DSc. Mayron César de Oliveira Moreira
Orientador

LAVRAS – MG

2022

ISABELA MARA DO NASCIMENTO

DESENVOLVIMENTO DE UM DASHBOARD PARA A VISUALIZAÇÃO DE CARDÁPIOS EM UM RESTAURANTE UNIVERSITÁRIO

Relatório técnico apresentado à Universidade Federal de Lavras como parte das exigências do curso de Sistemas de Informação, para a obtenção do título de Bacharel

APROVADA em 09 de Setembro de 2022.

Prof. DSc. Ramon Gomes Costa UFLA
Prof. DSc. Renata Teles Moreira UFLA

gov.br
Documento assinado digitalmente
MAYRON CESAR DE OLIVEIRA MOREIRA
Data: 22/09/2022 16:03:05-0300
Verifique em https://verificador.iti.br

Prof. DSc. Mayron César de Oliveira Moreira
Orientador

LAVRAS – MG
2022

RESUMO

O Gerador de cardápios utilizado neste trabalho foi desenvolvido por um grupo de pesquisadores da Universidade Federal de Lavras (UFLA), com o intuito de gerar cardápios diários para o Restaurante Universitário. Um conjunto de restrições deve ser respeitado, maximizando a variabilidade de preparações do cardápio em um dado horizonte de planejamento. A geração destes cardápios acontece através de um modelo matemático e uma heurística. A criação de um *dashboard* tem o propósito de gerar a visualização de dados, informações e estatísticas. Ao manipular qualquer tipo de dados que geram valor na computação, dependendo do volume dos dados e informações que serão geradas, é necessário criar uma forma de visualização organizada para entender os resultados. Um *dashboard* é útil ao processo de tomada de decisões, ao analisar estes resultados podemos decidir se a nossa solução é boa ou não, se os objetivos foram alcançados. As soluções obtidas pelo gerador de cardápios separa os cardápios em um arquivo de entrada por uma quantidade definida de dias, e o *dashboard* traz uma visualização simplificada com informações relevantes sobre os resultados.

Palavras-chave: Dashboard. Visualização. Estatísticas. Gerador de cardápios. Restaurante universitário.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Ao desenvolver o cardápio de um Restaurante Universitário, deve-se levar em consideração o valor nutricional das combinações de menus e também opções que atendam pessoas com restrições alimentares. Outro fator importante é a variedade de cardápios, servidos duas vezes ao dia, sete dias por semana, exceto fins de semana, onde é servida apenas uma refeição diária. Com o objetivo de criar uma ferramenta de apoio ao planejamento de cardápios, foi criado um modelo matemático e uma heurística construtiva que englobam as regras de negócio do contexto do Restaurante Universitário da Universidade Federal de Lavras. Com isso, pretende-se que a variedade e a nutrição estejam de acordo com a análise de especialistas na área.

Apesar do potencial dos resultados já produzidos pelos algoritmos, os dados gerados são apresentados de uma forma não amigável ao usuário. Quanto maior o número de cardápios gerados, mais difícil de ler, contabilizar e gerar informações sem a ajuda de um sistema ou ferramenta automatizada, que trate os dados brutos e gere conhecimento. Por este motivo, considerou-se a criação de um *dashboard* que por definição, são *displays* visuais de informações mais importantes para atingir um ou mais objetivos(FEW, 2006, p.26) para trazer uma melhor visualização ao usuário, além da extração dessas informações. O objetivo é que as soluções obtidas pelos algoritmos sejam tratadas e exibidas para que as tomadas de decisões sejam feitas com clareza e segurança. *Dashboards*, geralmente, são criados através de uma série de análises de dados e usa métricas chave que importam para as tomadas de decisão (HOWSON, 2008). No *dashboard* utilizado neste trabalho os menus foram dispostos em tabelas. Além disso, estatísticas sobre o planejamento dos cardápios são exibidas através de gráficos e informações sobre dados do cardápio.

Para desenvolver a interface da aplicação, foi utilizada uma solução *web*, onde não é necessário qualquer tipo de instalação de *software*. Com acesso à internet, é possível enviar arquivos no formato *Javascript Object Notation* (JSON) para uma *Application Programming Interface* (API) atuando no *back end* utilizando *NestJs*, enquanto o *front end* é composto por uma página desenvolvida usando *ReactJs* com a tabela de cardápios, dados sobre a quantidade de refeições diárias e gráficos com a quantidade dos tipos de preparações e alimentos que mais aparecem. Devido à geração eficiente e facilitada dos gráficos e tabela de menus diários, foi possível analisar os resultados pela profissional de nutrição encarregada de verificar as soluções geradas, e decidir a qualidade da alimentação sugerida e o que precisa de ser melhorado.

2 TECNOLOGIAS UTILIZADAS

2.1 Back end

O *Back end* da solução diz respeito a parte do desenvolvimento que lida com a infraestrura, servidor, APIs (*Application Programming Interface*), persistência de dados, etc., é o componente da programação que não está acessível ao usuário. A seguir, apresenta-se cada tecnologia utilizada no *back end* do *dashboard*.

2.1.1 NestJs

O *NestJs*[1] é um *framework* que utliza a linguagem *Typescript* ou *Javascript*, preparado para criar APIs aplicando *frameworks* comuns no *NodeJs*, como o *Express/Fastify*.

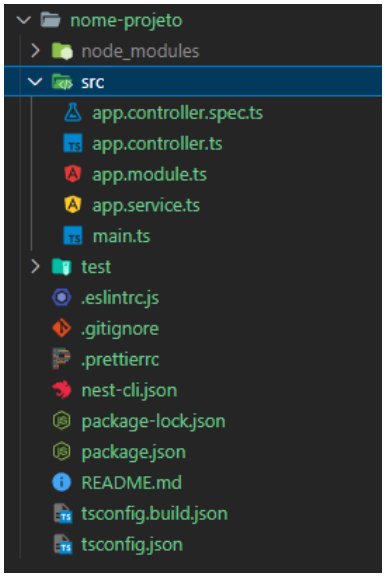
Para criar um projeto utilizando o *NestJS* é utlizado o *Node Package Manager* (NPM):

Figura 2.1 – Comandos para criar uma aplicação utilizando NestJs

```
$ npm i -g @nestjs/cli
$ nest new nome-projeto
```

Fonte:https://docs.nestjs.com/first-steps

Figura 2.2 – Estrutura inicial de um projeto utilizando NestJs



Fonte: Autora

Na Figura 2.1 são apresentados os comandos para criar um projeto utilizando o *NestJs* e na Figura 2.2 é mostrada a estrutura do projeto logo após sua criação.

[1] https://docs.nestjs.com/

2.1.1.1 NodeJs

O *NodeJs*[2] é um ambiente de execução *JavaScript* assíncrono e orientado a eventos, projetado para desenvolvimento de aplicações escaláveis de rede.

2.1.1.2 Typescript

O *Typescript*[3] é uma linguagem desenvolvida como um *superset* do *Javascript*. Ela contém todas funcionalidades do *Javascript*, mas com especificidades do *Typescript* como, tipagem estática, verificação estática de tipo, orientação a objetos. O *Typescript* adiciona uma melhor estruturação de código e prevenção de falhas ao *Javascript*. Ao compilar um código escrito em *Typescript*, o código é transpilado para *Javascript*.

O *Typescript* pode ser instalado via NPM em uma aplicação *node*:

Figura 2.3 – Comandos para instalar o Typescript em uma aplicação node

```
$ npm install typescript
ou
$ npm install typescript --save-dev
```

Fonte: https://www.typescriptlang.org/download

O *Typescript* também pode ser instalado via NPM globalmente:

Figura 2.4 – Comandos para instalar o Typescript globalmente

```
$ npm install -g typescript
```

Fonte: https://www.typescriptlang.org/download

Na Figura 2.3 são mostrados os comandos para instalar o *typescript* em uma apliação node, já na Figura 2.4 é mostrado como instalar o *typescript* globalmente através do npm.

[2] https://nodejs.org/pt-br/about/
[3] https://www.typescriptlang.org/

Figura 2.5 – Trecho de código em Typescript - exemplo-funcao.ts

```
function getLength(obj: string | string[]) {
    return obj.length;
}

console.log(getLength("Teste"))
```

Fonte: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

Figura 2.6 – Comando para compilar e transpilar o código em Typescript para javascript

```
$ tsc exemplo-funcao.ts
```

Fonte: Autora

Na Figura 2.5 é apresentado um trecho de código utilizando *Typescript*, onde temos uma função que recebe um parâmetro do tipo *string* ou um vetor de *strings* e retorna o tamanho dessa *string* ou vetor.

Figura 2.7 – Trecho de código transpilado de um código Typescript

```
function getLength(obj) {
    return obj.length;
}
console.log(getLength("Teste"));
```

Fonte: Autora

Após utilizar o comando da Figura 2.6 o código em *Typescript* é transpilado para *javascript* como demonstrado na Figura 2.7. No *Typescript* , caso seja passada por parâmetro uma variável que não seja do tipo *string*, o erro é acusado, devido à tipagem estática, diferente do que acontece no *Javascript* que apenas irá mostrar o resultado *undefined*.

2.1.1.3 Express

O *Express*[4] é um *framework web* minimalista para *NodeJS*, contendo métodos utilitários HTTP e *middleware* para criar APIs de forma rápida.

na Figura 2.8 é mostrado como instalar o *Express* em um projeto node:

[4] https://expressjs.com/pt-br/

Figura 2.8 – Comando para instalar o express em um projeto node

```
$ npm install express --save
```

Fonte: https://expressjs.com/pt-br/starter/installing.html

2.2 Front end

O *Front end* corresponde a parte de interface gráfica das aplicações.

2.2.1 Reactjs

O *ReactJs*[5] é uma biblioteca *Javascript* para criar *interfaces* de usuário. Para criar um projeto utilizando o *ReactJs* é necessário ter o *NodeJs* instalado.

Na Figura 2.9 é apresentado o comando para criar um projeto utilizando *ReactJs*:

Figura 2.9 – Comandos criar um projeto Reactjs

```
$ npx create-react-app my-app
```

Fonte: https://reactjs.org/docs/create-a-new-react-app.html#create-react-app

2.2.1.1 Javascript

O *Javascript*[6] (*ECMAScript*) é uma linguagem que pode ser utilizada em aplicações *web*, sendo responsável por definir o comportamento das páginas, integração com o *back end* e utilização de bibliotecas com ferramentas e funcões necessárias à aplicação.

2.2.1.2 JSX

O *JSX*[7] é uma extensão de sintaxe para *JavaScript* utilizada dentro do *ReactJs*. Possui um formato similar ao *XML*, integrando em apenas uma página a estruturacão de uma página ou componente, *Javascript* e estilização da página. A seguir, na Figura 2.10 um trecho de código utilizando o JSX.

[5] https://pt-br.reactjs.org/
[6] https://developer.mozilla.org/pt-BR/docs/Web/JavaScript
[7] https://pt-br.reactjs.org/docs/introducing-jsx.html

Figura 2.10 – Trecho de código utilizado em uma apliação ReactJs

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

Fonte: https://reactjs.org/docs/create-a-new-react-app.html#create-react-app

2.2.1.3 Cascading Style Sheets (CSS) e Bootstrap

O *CSS* é uma linguagem utilizada para estilizar páginas web. Já o *Bootstrap*[8] é um pacote de *features* para estilização *front end*, que cria classes com pré configurações de propriedades de *CSS*, além de oferecer ícones e templates para a estilização do *front end*.

2.2.1.4 React-Chartjs

O *ChartJs*[9] para o *ReactJs* é uma biblioteca *Javascript* preparada para criação de gráficos.

Na Figura 2.11 é mostrado como instalar o *React-ChartJs* em um projeto node:

[8] https://getbootstrap.com/
[9] https://react-chartjs-2.js.org/

Figura 2.11 – Comando para instalar o react-charjs-2 em um projeto ReactJs

```
$ npm i react-chartjs-2 chart.js
```

Fonte: https://www.npmjs.com/package/react-chartjs-2

2.3 Infraestrutura e deploy

O *dashboard* é uma aplicação web e foi implantada em um servidor *Virtual Private Server* (VPS) linux para estar disponível via *Hypertext Transfer Protocol*(http).

2.3.1 Git e Github

O Git[10] é um sistema de controle de versão de projetos e o Github[11] é um gerenciador de repositórios que serve como *host* de projetos oferendo também ferramentas de construção, automação, distruibuição, documentação e *deploy*.

2.3.2 PM2

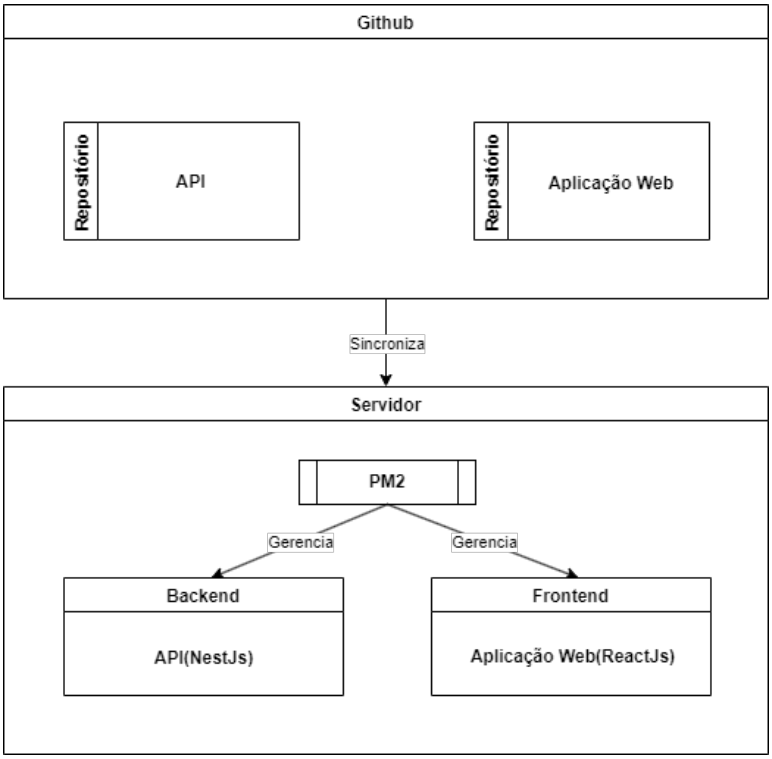
O PM2[12] é um gerenciador de processos *daemon* que mantém aplicações disponíveis no servidor.

2.4 Como as tecnologias são utilizadas

Os repositórios com os códigos fonte estão disponíveis no github, onde é possível manter um controle de versões da solução. No servidor as aplicações são atualizadas de acordo com os repositórios do github e para manter as aplicações disponíveis é utilizado o PM2, como mostrado na Figura 2.12.

[10] https://git-scm.com/
[11] https://github.com/
[12] https://pm2.keymetrics.io/

Figura 2.12 – Utilização das tecnologias



Fonte: Autora

3 DESCRIÇÃO DO TRABALHO

O gerador de cardápios base para criação do *dashboard* foi feito para atender necessidades do Restaurante Universitário (RU) da UFLA. Os cardápios são gerados através de um modelo matemático ou uma heurística. O RU serve refeições duas vezes por dia de segunda à sexta, e uma refeição por dia nos fins de semana. Tais refeições precisam atender critérios a respeito da variedade de cardápios e informações nutricionais. O planejamento obtido pelo gerador de cardápios traz os cardápios diários com o menu de cada refeição servida, no almoço e no jantar, durante um número determinado de dias.

Os dados brutos originados do gerador de cardápios não são amigáveis ao usuário, gerando um problema em que o profissional responsável por avaliar os cardápios teria que fazer um trabalho manual avaliando cada cardápio individualmente. Dependendo da quantidade de cardápios obtidos, o trabalho se torna maçante e propenso a erros de avaliação.

Os alimentos e preparações são separados nas seguintes categorias: opção vegetariana, salada, opção proteica e guarnição. Cada menu é uma combinação de alimentos e preparações correspondentes a cada uma dessas classificações. Porém, existem condições para ser atendidas em relação a quantidade de preparações presentes em cada menu, e o número de vezes que cada preparação do menu aparece em um determinado intervalo de tempo.

O modelo matemático e a heurística foram desenvolvidos levando em conta critérios e restrições como:

- A quantidade que cada preparação ou alimento pode ser servida por dia, semana e em cada refeição, por exemplo: oferecer ao menos uma opção vegana por refeição, oferecer duas opções de salada por refeição, opções proteicas variadas por refeição com tipos de proteínas distintas (bovina, suína, aves e peixes), restringir a quantidade de vezes que preparações ou alimentos de alto custo podem ser utilizadas, dentre várias outras restrições.

- Alguns alimentos ou prepações não podem ser servidos juntos.

- Existe um número ideal da quantidade de cada alimento ou preparação servidos semanalmente.

- Alguns alimentos ou preparações sempre aparecem juntos.

- Também é necessário pensar sobre a forma de preparo das refeições levando em conta a utilização de utensílios e infraestrutura da cozinha.

Ou seja, o número de restrições e critérios trazem uma complexidade na geração e análise dos cardápios.

A forma como os dados são apresentados nas soluções é costumizável. É possível, por exemplo, gerar os dados em planilhas com formato *".csv"*, *".xml"* e *".json"*. A modelagem das informações contidas no arquivo de saída dependem do tipo de dados necessários aos usuários do gerador de cardápios. O *dashboard* foi desenhado para atender a necessidade de mostrar informações sobre o resultado criado pelo gerador de cardápios. Para tanto, foi definido que os cardápios fossem apresentados visualmente de uma forma amigável ao usuário. Outra necessidade é apresentar a quantidade de vezes que alimentos e preparações aparecem, a fim de atender restrições definidas em relação a variedade e valores nutricionais.

4 DISCUSSÃO E ANÁLISE DO PROBLEMA

Com a geração dos cardápios através do modelo matemático ou heurística surgiram alguns questões, como:

- Qual a melhor forma de apresentar os resultados gerados para o usuário final?
- Como extrair informações relevantes que auxiliem na análise da qualidade dos resultados gerados?

O processo de levantamento e análise de requisitos foi feito em conjunto com um *stakeholder* do projeto do gerador de cardápios onde foram seguidos os seguintes passos:

1. Apresentação sobre o gerador de cardápios e seu funcionamento.
2. Discussão do problema acerca da necessidade de criar uma ferramenta de apresentação dos resultados gerados.
3. Decisão sobre qual ferramenta desenvolver para resolver o problema.

Após analisar o problema foi definida a criação de um *dashboard* que apresenta uma solução viável para o perfil do problema apresentado, pois *dashboards* são ferramentas de visualização de métricas e informações.

5 IMPLEMENTAÇÃO

A implementação do *dashboard* considerou, inicialmente, uma codificação que prezasse por escalabilidade e mantenabilidade, pensando na possibilidade de adicionar ou remover funcionalidades de acordo com as necessidades. Também foi levado em conta atender as necessidades dos usuários, fazendo um *design* simples e voltado para a solução do problema.

Para a criação do *dashboard*, existiam algumas opções a se considerar: desenvolver uma aplicação desktop multiplataforma para Linux ou Windows, ou desenvolver uma aplicação *web* disponível em um servidor. Foi escolhida a opção *web*, pois traz a possibilidade de acesso de qualquer lugar com internet sem a necessidade de instalar a aplicação em uma máquina localmente.

A aplicação está dividida em *back end* e *front end* que foram desenvolvidos de forma separada. Tem-se uma API que lida com o processamento do arquivo de entrada e disponibilização dos dados em formato *JSON*, através de requisições *POST* e *GET*. O *front end* recebe os dados do *back end*, transformando-os em informações exibidas ao usuário.

5.1 Decisões sobre arquitetura e tecnologias

Por se tratar de uma aplicação voltada para um problema muito específico, de escopo fechado e pequeno, houve a escolha de utilizar uma implementação *web*, pensando na facilidade de uso e disponibilidade. A escolha pelas tecnologias *NestJs* e *ReactJs* se deram pela facilidade de criação das estruturas de *back* e *front end* e também pela escalabilidade que as duas tecnologias oferecem, é possível adicionar e remover *features* sem se preocupar com detalhes de configuração. A manipulação de dados, assim como a geração de tabelas e gráficos ficam mais simples graças a quantidade de bibliotecas disponíveis para aplicacações que utilizam *Javascript/Typescript* .

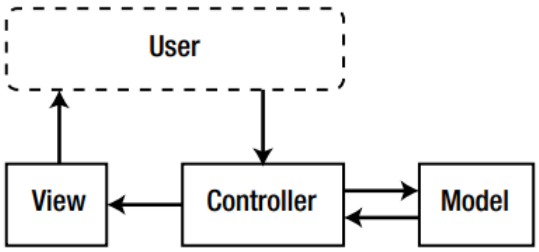
5.2 Persistência de dados

A definição do formato da persistência dos dados foi baseada nas necessidades que o projeto demanda. Não foi necessário utilizar um banco de dados, pois o *dashboard* serve como ferramenta de visualização de informações sobre as soluções criadas no gerador de cardápios, sem precisar persistir esses dados para consultas futuras. Um arquivo de entrada é salvo em uma pasta no *back end*, sendo atualizado toda vez que um arquivo novo é enviado.

5.3 Back end

O *back end* foi desenvolvido utilizando o padrão de arquitetura Model View Controller (*MVC*). O padrão MVC divide responsabilidades em três categorias principais: o modelo do domínio de aplicação principal, a apresentação de dados nesse modelo e interação do usuário (KRASNER; POPE, 1988).

Figura 5.1 – Arquitetura MVC

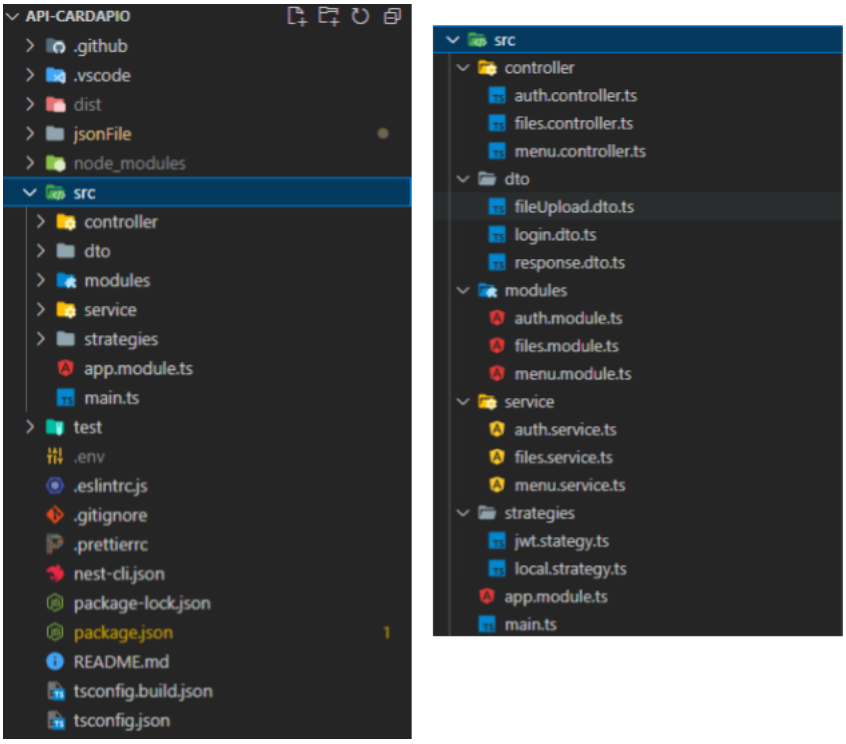


Fonte: Livro - Pro PHP MVC - (PITT, 2012, p.2)

Como apresentado na Figura 5.1 na arquitetura MVC as requisições do usuário são tratadas na camada *controller*, os dados ficam na camada *model* e as respostas ao usuário são de reponsabilidadade da camada de visulização(*view*).

A utilização dessa arquitetura na API foi escolhida pensando na mantenabilidade e escalabilidade da aplicação, já que apresenta uma organização conhecida e mais intuitiva. Na Figura 5.2 é exibida a estrutura da API desenvolvida no projeto:

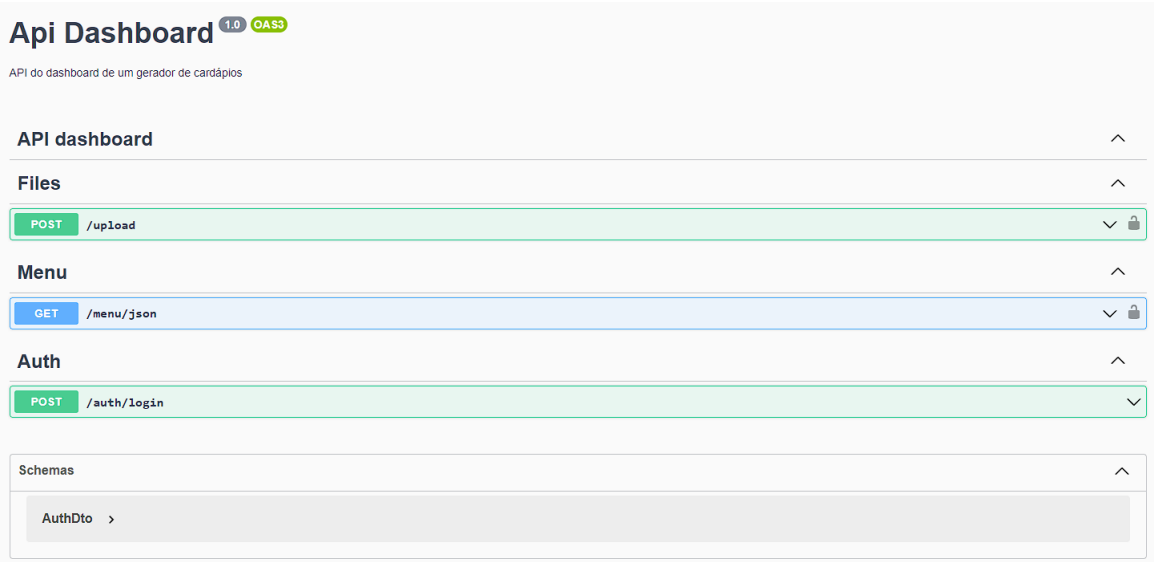
Figura 5.2 – Estrutura da API



Fonte: Autora

Foi criada uma API utilizando o framework *NestJs*, devido à facilidade de gerar um projeto estruturado pronto para uso com um nível mais alto de abstração, que elimina a necessidade de implementar detalhes de configuração requeridas em projetos *NodeJs*. A API recebe e processa o arquivo de entrada no formato *JSON* e disponibiliza o arquivo através de uma requisição *GET*. O tratamento dos dados é feito no *front end* para reduzir a quantidade de requisições já que não há uma alta quantidade de dados e filtros na tabela de visualização dos cardápios.

Figura 5.3 – Interface da API



Fonte: Autora

Na Figura 5.3 é mostrada uma *interface* com as rotas presentes na API.

A API possui três rotas como mostra o quadro:

Quadro 5.1 – Quadro de requisições da API

Método	Rota
POST	{host}/auth/login
POST	{host}/upload
GET	{host}/menu/json

Cada requisição é responsável por uma operação listada a seguir:

1. *POST* /auth/login - Requisição para autenticação dos usuários:

```
curl --location --request POST 'http://31.220.57.9:3000/auth/login' \
--header 'Content-Type: application/json' \
--data-raw '{
  "user": "usuarioTeste",
  "password": "senha123"
}'
```

A requisição retorna como resposta um *JSON Web Token*[1] (JWT), o qual é um padrão aberto (RFC 7519)[2] que define uma forma compacta e independente de transmitir in-

[1] https://jwt.io/introduction
[2] https://www.rfc-editor.org/rfc/rfc7519

formações entre partes com segurança como um objeto JSON. O JWT recebido como resposta da requisição é utilizado para autenticação ao fazer requisições ao *back end*:

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
    .eyJzdWIiOiJ1c3VhcmlvVGVzdGUiLCJpYXQiOjE2NjE2OTk0NzgsImV4cCI6MTY2MTcwMzA3OH0
    .MVW5a7W8xp97RUFh6oMTRzxs9aU3I3vVgsMaBpzc0GE"
}
```

2. *POST* /upload - Requisição para envio do arquivo de entrada:

```
curl --location --request POST 'http://31.220.57.9:3000/upload' \
--header 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
.eyJzdWIiOiJ1c3VhcmlvVGVzdGUiLCJpYXQiOjE2NjE2OTk0NzgsImV4cCI6MTY2MTcwMzA3OH0
.MVW5a7W8xp97RUFh6oMTRzxs9aU3I3vVgsMaBpzc0GE' \
--form 'file=@"/C:/Users/isabe/TCC/resultados_classificacao.json"'
```

Ao enviar um arquivo no formato *.json* que contenha o formato apresentado na Figura 5.4, a resposta a seguir é retornada:

```
{
    "status": 200,
    "message": "Upload Realizado"
}
```

Caso contrário, uma resposta de erro é retornada da requisição, no exemplo de resposta a seguir, é exibido o resultado de uma requisição, onde o formato do arquivo enviado é diferente de *.json*:

```
{
    "status": 400,
    "message": "Arquivo não possui o formato json"
}
```

3. *GET* /menu/json - Requisição para receber dados do arquivo:

```
curl --location --request GET 'http://31.220.57.9:3000/menu/json' \
--header 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
```

```
.eyJzdWIiOiJ1c3VhcmlvVGVzdGUiLCJpYXQiOjE2NjE2OTk0NzgsImV4cCI6MTY2MTcwMzA3OH0
.MVW5a7W8xp97RUFh6oMTRzxs9aU3I3vVgsMaBpzc0GE'
```

A resposta obtida na requisição /menu/json corresponde à Figura 5.4 e traz os cardápios separados por semana, dia, almoço ou janta.

Figura 5.4 – Trecho do resultado obtido pelo gerador de cardápios.

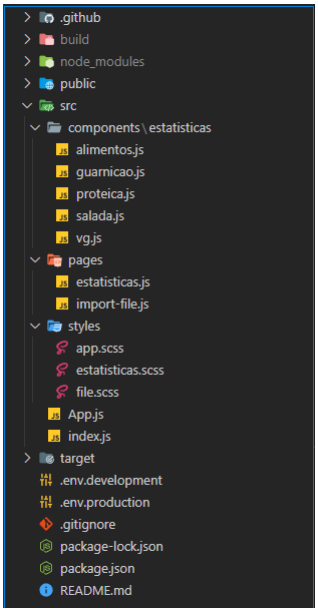

```
{
"semana1": {
    "dia1": {
        "almoco": [
            ["carne-com-batata", "proteica"], ["escarola", "salada"],
            ["chicoria", "salada"],
            ["torta-de-pts-com-brocolis-e-milho, "vg"],
            ["pts-com-legumes", "vg"]
        ],
        "janta": [
            ["isca-de-frango-com-ervilha", "proteica"],
            ["costelinha-assada-ao-molho-de-abacaxi", "proteica"],
            ["cuscuz", "guarnicao"], [ "acelga", salada"],
            ["repolho-branco", "salada"],
            ["pts-com-repolho-e-cenoura", "vg"],
            ["macarrao-de-forno-com-pts-e-milho", "vg"]
        ]
        }
    }
}
```


Fonte: Autora

5.4 Front end e comunicação com o Back end

O *front end* foi desenvolvido em *ReatJs* que oferece um estrutura onde é possível criar componentes reutilizáveis, além de possuir um grande número de bibliotecas para auxiliar na criação de páginas dinâmicas e responsivas. Uma página estática não atenderia a necessidade de processar um arquivo de entrada para transformar dados em tabelas e gráficos. Na Figura 5.5 é mostrada a estrutura do *front end* da aplicação:

Figura 5.5 – Estrutura do Front end



Fonte: Autora

Na estrutura, a pasta principal src contém:

1. A pasta *components* recebe componentes definidos pelo desenvolvedor, qualquer estrutura a ser utilizada dentro das páginas da aplicação pode ser componentizada, no *dashboard* os gráficos utilizados são divididos em componentes separados.

2. A pasta *pages* contém as páginas presentes da aplicação.

3. A pasta *styles* contém os arquivos de estilização em css da aplicação.

4. O arquivo App.js que possui o menu principal que é estático e funciona com *container* para todas a páginas.

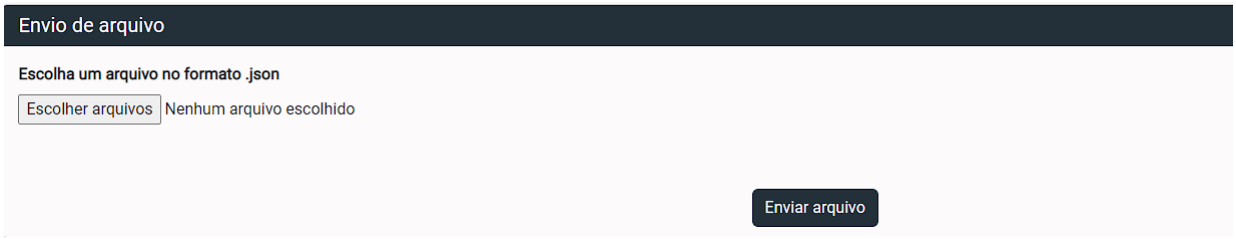
5. O arquivo index.js que tem algumas configurações iniciais do react para criação e montagem das páginas.

O arquivo de entrada para geração do *dashboard* mostrado na Figura 4.3 utiliza a estrutura chave, valor no formato *JSON* mostrada na Figura 5.4 e possui as seguintes informações: semana, dia, preparações e alimentos divididos entre almoço e jantar com suas respectivas classificações.

Ao abrir a aplicação no *browser*, na página inicial é encontrada uma seção para enviar o arquivo de entrada no formato *.json*. O arquivo é enviado ao backend através de uma requisição

POST, é processado pela API e fica disponível para consulta até que seja enviado outro arquivo ao *back end*.

Figura 5.6 – Envio de arquivo

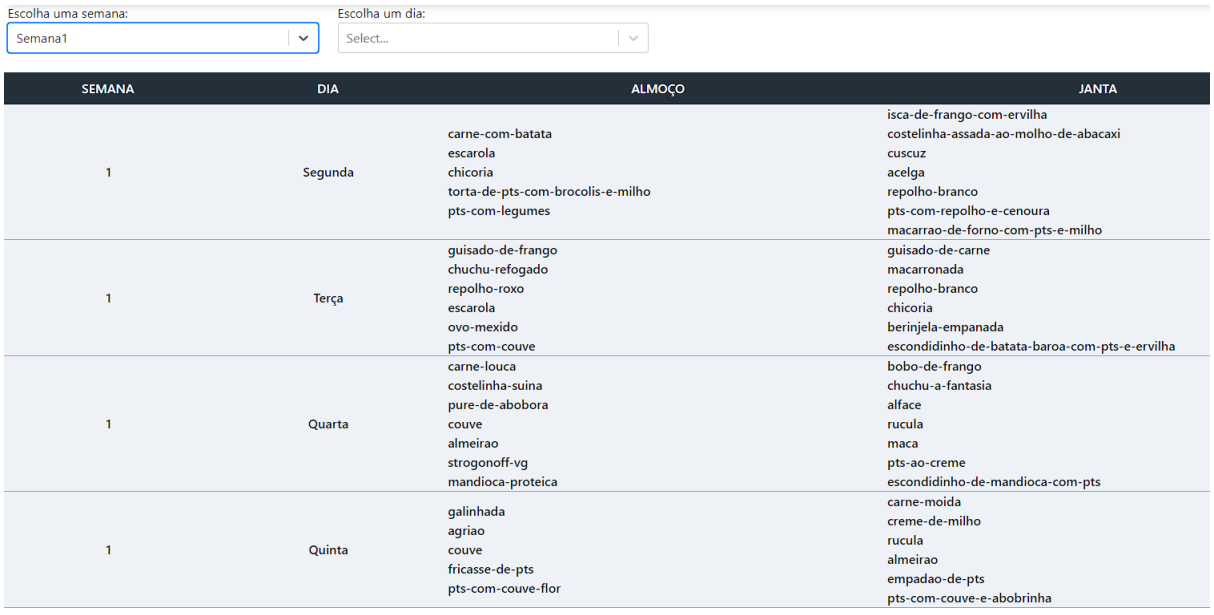


Fonte: Autora

Na Figura 5.6 é apresentada a estrutura responsável pelo envio do arquivo de entrada da aplicação, onde o usuário tem a opção de escolher um arquivo .json e envia-lo.

Utilizando a requisição *GET* /menu/json, os dados são recebidos no *front end* e os cardápios são exibidos em formato de tabela na página inicial, tendo como filtros semana e dia, como é mostrado na Figura 5.7.

Figura 5.7 – Tabela de cardápios



SEMANA	DIA	ALMOÇO	JANTA
1	Segunda	carne-com-batata escarola chicoria torta-de-pts-com-brocolis-e-milho pts-com-legumes	isca-de-frango-com-ervilha costelinha-assada-ao-molho-de-abacaxi cuscuz acelga repolho-branco pts-com-repolho-e-cenoura macarrao-de-forno-com-pts-e-milho
1	Terça	guisado-de-frango chuchu-refogado repolho-roxo escarola ovo-mexido pts-com-couve	guisado-de-carne macarronada repolho-branco chicoria berinjela-empanada escondidinho-de-batata-baroa-com-pts-e-ervilha
1	Quarta	carne-louca costelinha-suina pure-de-abobora couve almeirao strogonoff-vg mandioca-proteica	bobo-de-frango chuchu-a-fantasia alface rucula maca pts-ao-creme escondidinho-de-mandioca-com-pts
1	Quinta	galinhada agriao couve fricasse-de-pts pts-com-couve-flor	carne-moida creme-de-milho rucula almeirao empadao-de-pts pts-com-couve-e-abobrinha

Fonte: Autora

Na página de estatísticas, há uma primeira seção com textos indicando alguns números sobre os cardápios, como: o total de refeições, semanas, dias, se algum menu se repete, caso haja repetição é exibido o tempo mínimo e o tempo médio para as repetições ocorrerem, como mostrado na Figura 5.8. Estes números são importantes para analisar se as restrições definidas

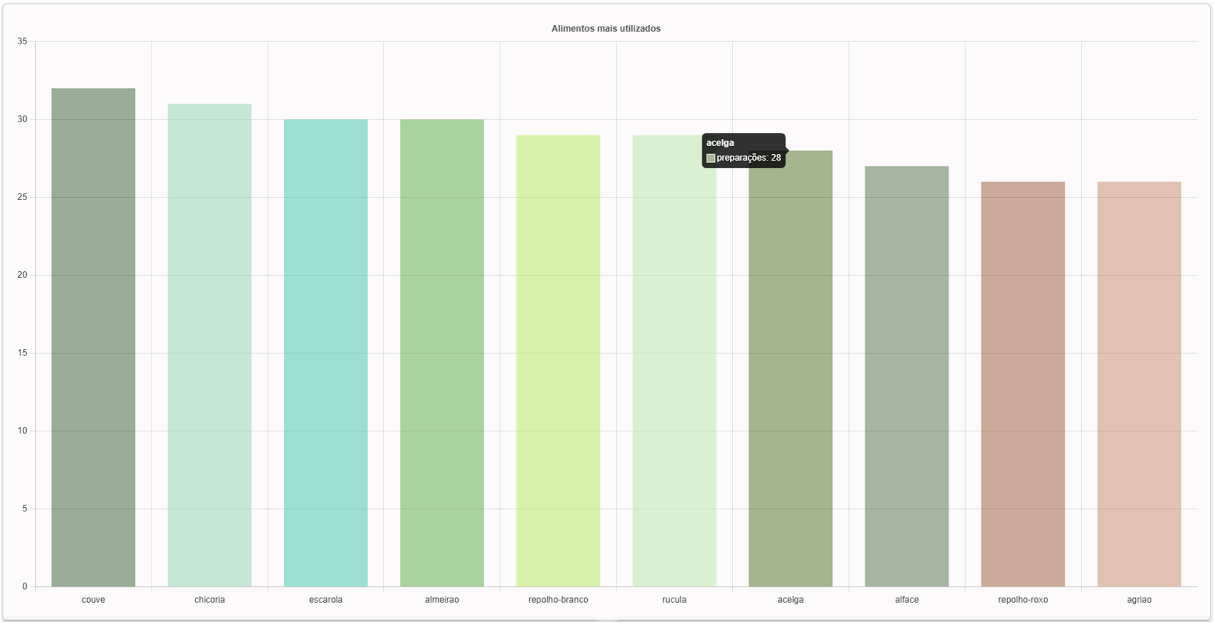
estão sendo atendidas, o conjunto de cardápios diários com doze semanas como o do exemplo cobre cerca de 23% do ano, com um total de 144 menus diferentes.

Figura 5.8 – Dados textuais sobre os cardápios

Fonte: Autora

Logo após os dados textuais, são apresentados gráficos com o total de alimentos e preparações mais utilizadas no geral e por classificação. Estes gráficos são gerados através de uma biblioteca chamada *ChartJs*. Para gerar tais gráficos utiliza-se os dados recebidos via requisição à API. Os desenhos são gerados conforme as configurações definidas pelo usuário, na Figura 5.9 é mostrado um gráfico em formato de barras verticais com dez preparações mais utilizadas nos cardápios e a quantidade de vezes que cada uma delas aparece.

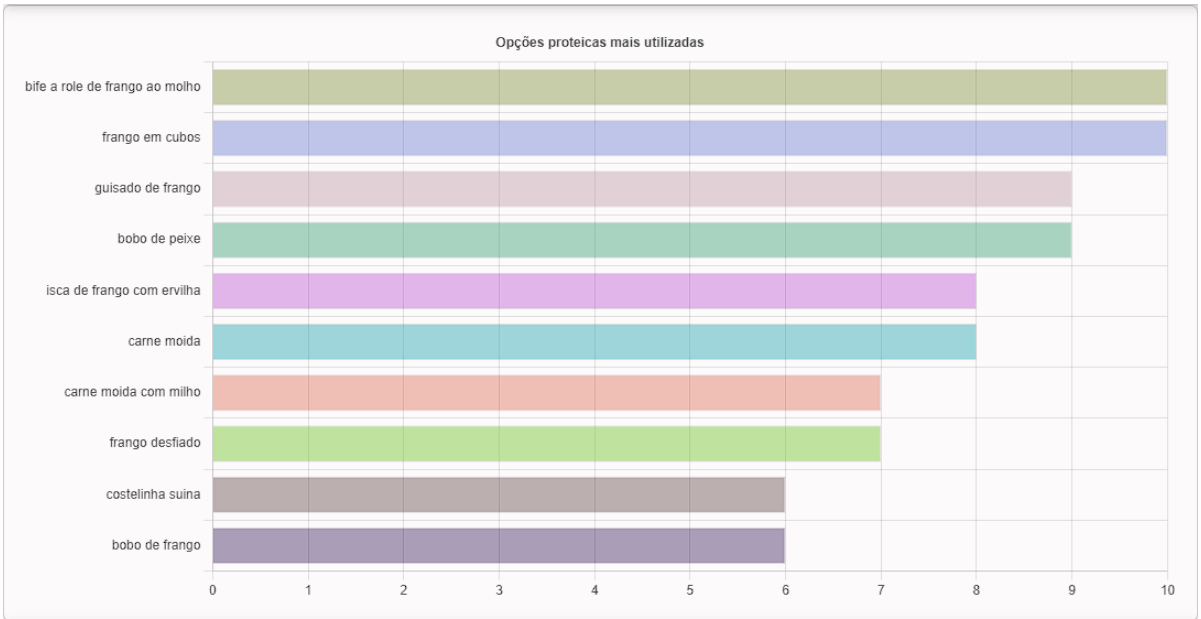
Figura 5.9 – Gráfico de preparações mais utilizadas



Fonte: Autora

Na Figura 5.10 o gráfico apresentado no formato de barras horizontais mostra dez preparações preteicas mais utilizadas nos cardápios e a quantidade de vezes que cada uma delas aparece:

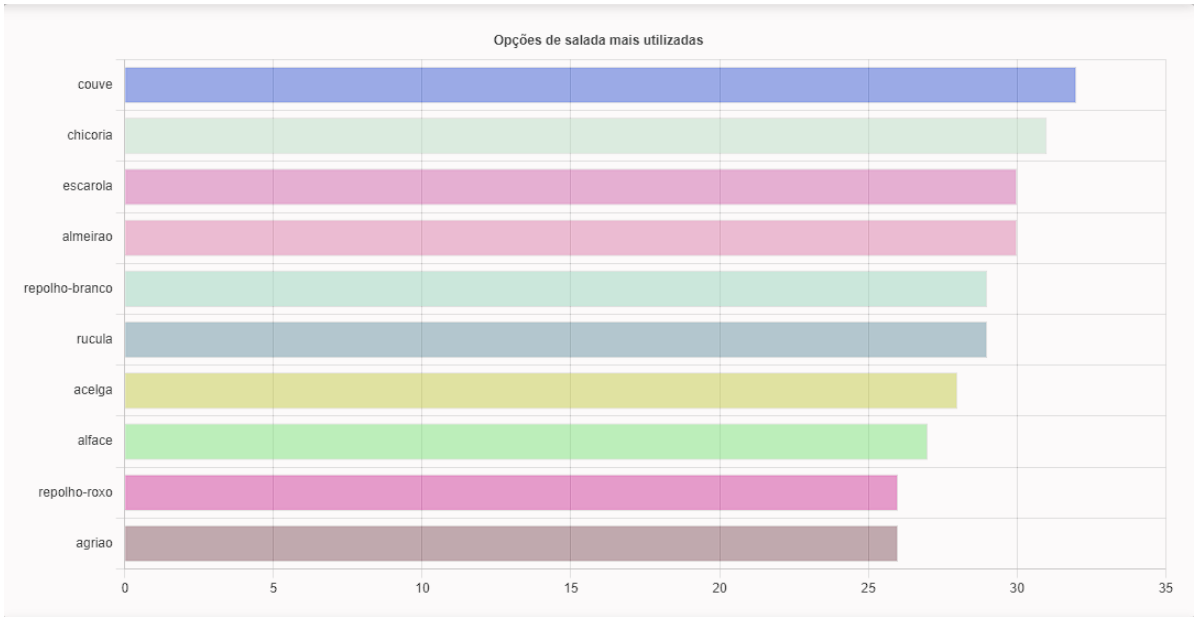
Figura 5.10 – Gráfico de opções proteicas mais utilizadas



Fonte: Autora

Na Figura 5.11 o gráfico apresentado no formato de barras horizontais mostra dez opções de salada mais utilizadas nos cardápios e a quantidade de vezes que cada uma delas aparece:

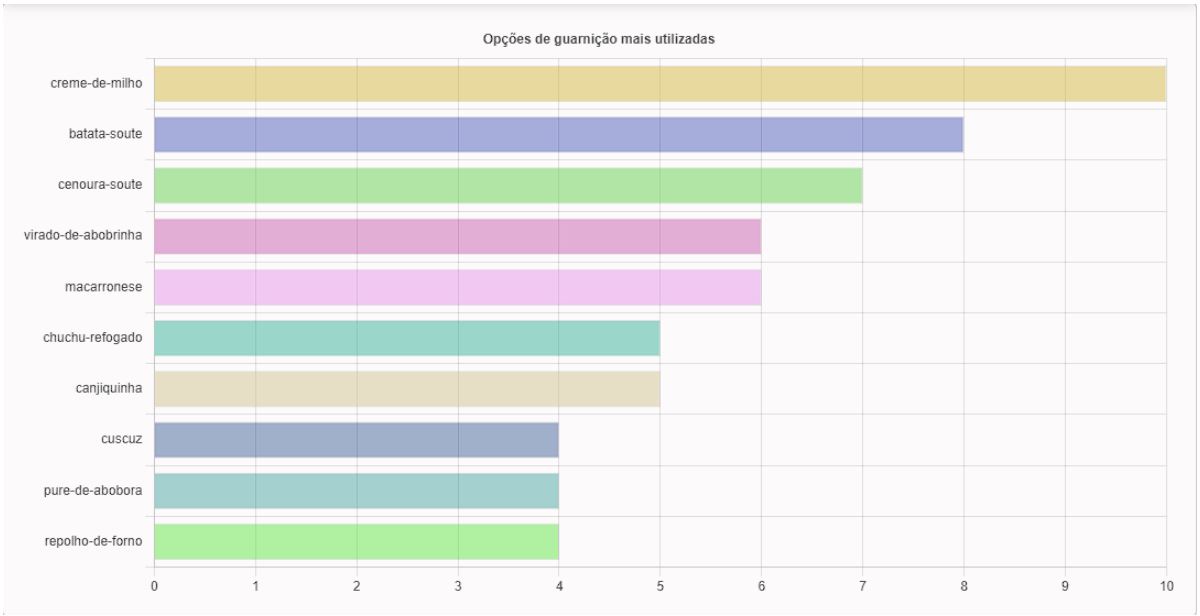
Figura 5.11 – Gráfico de opções de saladas mais utilizadas



Fonte: Autora

Na Figura 5.12 o gráfico apresentado no formato de barras horizontais mostra dez guarnições mais utilizadas nos cardápios e a quantidade de vezes que cada uma delas aparece:

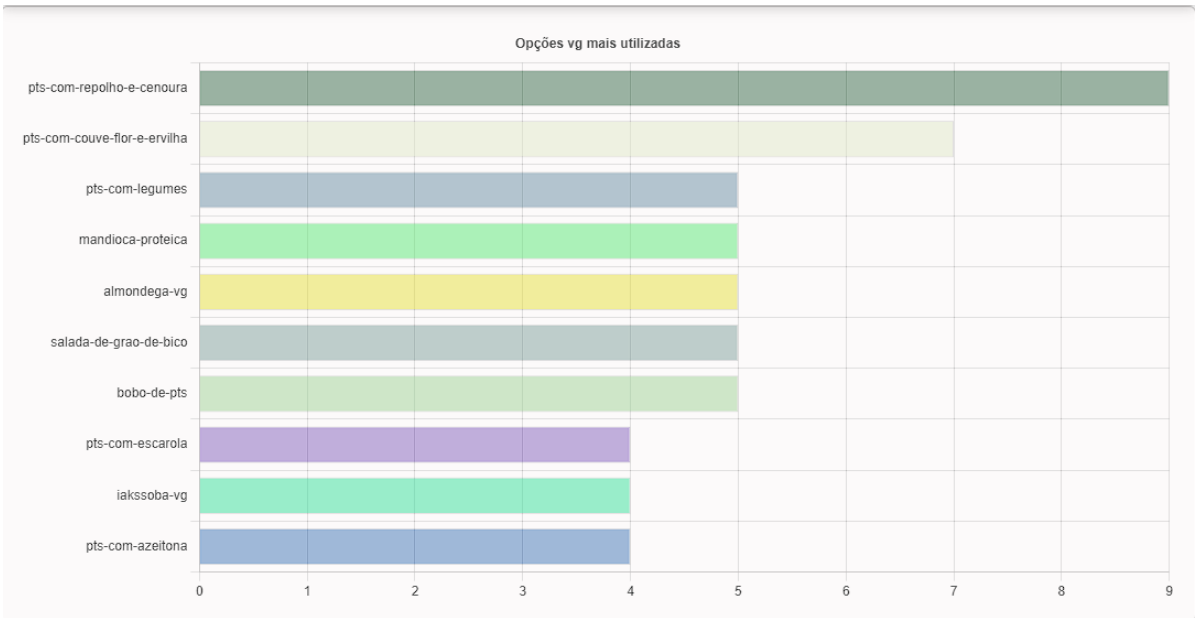
Figura 5.12 – Gráfico de guarnições mais utilizadas



Fonte: Autora

Na Figura 5.12 o gráfico apresentado no formato de barras horizontais mostra as dez opções vegetarianas mais utilizadas nos cardápios e a quantidade de vezes que cada uma delas aparece:

Figura 5.13 – Gráfico de opções vegetarianas mais utilizadas



Fonte: Autora

5.5 Implantação

Tanto a API, quanto o *front end* do *dashboard* foram implantados em um servidor VPS utilizando o git para controle de versão e atualização do código-fonte no servidor. O gerenciador que mantém a aplicação *online* é o PM2, que possui um controle de *logs* e monitoramento das aplicações.

Figura 5.14 – Aplicações no servidor

id	name	namespace	version	mode	pid	uptime	↺	status	cpu	mem	user	watching
24	api-cardapio	default	N/A	fork	18104	44h	2	online	0%	51.6mb	root	disabled
19	dashboard	default	N/A	fork	2702	8D	298	online	0%	18.9mb	root	disabled

Fonte: Autora

Na Figura 5.14 são mostradas informações sobre às aplicações presentes no servidor como, *status*, nome, tempo disponível *online*, entre outros, através do PM2.

5.6 Resultados obtidos

O desenvolvimento do *dashboard* fornece uma ferramenta para profissionais de nutrição responsáveis por avaliar a solução obtida pelo gerador de cardápios. Através dessa ferramenta, pode-se definir em tempo hábil a qualidade da solução gerada, identificando problemas em relação à variedade, repetição e composição de cada cardápio. O *dashboard* atende a sua principal função que é oferecer métricas e informações capazes de apoiar as tomadas de decisões, podendo assim, traçar processos de melhoria ou aprovação das soluções implementadas.

6 CONCLUSÃO

O *dashboard* é uma ferramenta poderosa que traz métricas e resultados tratados e abstraídos que ajudam *stakeholders* acompanharem o andamento, resultados ou desempenho de processos, tarefas e ferramentas. O valor agregado que um *dashboard* pode trazer é o acompanhamento eficiente de dados, informações e estatísticas, que auxiliem no processo de tomada de decisões, reduzindo consideravelmente alguns trabalhos possivelmente manuais e decentralizados. A ferramenta ajuda, também, a identificar problemas, ou resultados indesejados que sem tal recurso poderiam não ser considerados.

Criar um cardápio diverso e balanceado, que respeite diversas restrições e critérios de forma otimizada, se torna uma tarefa muito mais eficiente utilizando um sistema computadorizado, especialmente agregando valor ao gerador de cardápios usando um *dashboard*. Mostrar os resultados graficamente organizados traz a comodidade de fazer análises mais rápidas e assertivas. Pensando em contextos maiores, onde há um fluxo de dados muito grande, a utilização de *dashboards* pode significar um ganho considerável de resultados melhores e alcance de objetivos.

Para criar o *dashboard* do gerador de cardápios, foi necessário utilizar o conhecimento que engloba diversas áreas do curso de Sistemas de Informação como: Engenharia de *Software*, Desenvolvimento e Estrutura de Dados, Interação Humano-Computador, entre outros. O desenvolvimento do *dashboard* envolve tarefas como:

- Identificação e análise de requisitos.
- Entender o problema e modelar uma solução viável.
- Definir tecnologias e arquitetura da solução escolhida.
- Implementar a aplicação atuando nas áreas de desenvolvimento, incluindo:
  - *Design* e experiência do usuário.
  - *Front end*.
  - *Back end*.
  - *Deploy* das aplicações.

No início do desenvolvimento do trabalho, na fase de levantamento e análise de requisitos, o maior desafio encontrado foi definir todas funcionalidades a serem desenvolvidas. Na

parte de implementação, o maior desafio foi testar *frameworks* e ferramentas desconhecidos no desenvolvimento do *front end*, que demandava a inclusão de gráficos de forma dinâmica ao gerar informações sobre os cardápios. A tecnologia usada para desenvolver o *front-end* teve que ser alterada no desenvolvimento do trabalho, devido à incompatibilidade com a ferramenta de geração de gráficos.

Em resumo, os maiores desafios foram lidar com o ambiente de mudanças, de criar e desenvolver uma aplicação em um contexto real, participando de todas as etapas desde a concepção até a implantação, e aprender novas tecnologias que atendessem as necessidades da solução.

Ter a experiência de participar de todas as etapas do processo de criar e implementar uma solução, utilizando o conhecimento de vários campos dentro da área de tecnologia da informação, foi muito enriquecedor para à discente entender como é possível contribuir e agregar valor a processos em diversas áreas de atuação.

REFERÊNCIAS

FEW, S. **Information Dashboard Design: The Effective Visual Communication of Data**. [S.l.]: O'Reilly Media, Inc., 2006. ISBN 0596100167.

HOWSON, C. **Successful business intelligence : secrets to making BI a killer app**. 1st edition. ed. New York, NY: McGraw-Hill, 2008. ISBN 1-281-15188-2.

KRASNER, G. E.; POPE, S. T. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. **J. Object Oriented Program.**, SIGS Publications, Inc., USA, v. 1, n. 3, p. 26–49, aug 1988. ISSN 0896-8438.

PITT, C. **Pro PHP MVC**. 1st edition. ed. [S.l.]: Apress Berkeley, CA, 2012. ISBN 978-1-4302-4165-2.

RODRIGUES, G. S. Criação do sistema de banco de dados como apoio para cardápios automatizados de restaurantes universitários. 2022. Disponível em: <http://repositorio.ufla.br/jspui/handle/1/54974>.