Relatório de Estágio

Mateus Rocha

Curso Técnico Superior Profissional em
Desenvolvimento de Aplicações Informáticas

dez | 2022

GUARDA
POLI
TÉCNICO

Relatório de Estágio

Mateus Rocha

Instituto Politécnico da Guarda

Escola Superior de Tecnologia e Gestão

Desenvolvimento de Aplicações Informáticas

Estágio

Dezembro de 2022

Relatório de Estágio

Mateus Rocha
1705389

Relatório final de estágio

Orientado por: Noémio Dória

Instituto Politécnico da Guarda
Escola Superior de Tecnologia e Gestão
Desenvolvimento de Aplicações Informáticas
Estágio

Dezembro de 2022

Ficha Técnica

Estagiário

Nome: Mateus Rocha

Nº de Aluno: 1705389

Curso: Técnico Superior Profissional – Desenvolvimento de Aplicações Informáticas

Estabelecimento de Ensino

Escola Superior de Tecnologia e Gestão

Instituto Politécnico da Guarda

Morada: Avenida Dr. Francisco Sá Carneiro nº50, 6300-559 Guarda

Telefone: +351 271 220 164

Email: estg-geral@ipg.pt

Website: www.estg.ipg.pt

Entidade Acolhedora

Shore Spun Lda.

Área de Atividade: Consultoria na área de TI

Morada: Avenida Defensores de Chaves nº15, 7º Esq, 1000-109 Lisboa

Telefone: +351 216 045 002

Email: geral@shore.pt

Website: www.shore.pt

Duração do Estágio

Inicio: 07/03/2022

Final: 22/07/2022

Supervisor: André Cabeças

Docente Orientador

Nome: Noémio Dória

Plano de Estágio Curricular

No âmbito do estágio curricular do Curso Técnico Superior Profissional – Desenvolvimento de Aplicações Informáticas a realizar na empresa Shore Spun, foi estabelecido um plano de estágio.

O plano de ensino acordado com o supervisor do estágio e aceite pelo docente-orientador sugeria que eu ganharia experiência com produção de código, *back-end* com Java e C#, e *front-end* com Angular ou React. Para aplicações móveis, iria ser definido posteriormente se seria com sistemas nativos ou *cross-platform* em Flutter.

Resumo

Este relatório apresenta os resultados do estágio curricular de 750 horas do curso de Desenvolvimento de Aplicações Informáticas do Instituto Politécnico da Guarda, realizado entre 07 de Março de 2022 até 22 de Julho do mesmo ano. Foi realizado na empresa Shore Spun Lda, com o principal foco em desenvolvimento *Web front-end*, utilizando a *framework* Angular, que utiliza *TypeScript*, em conjunto com CSS, HTML e JavaScript. Após o contacto inicial com as ferramentas e um período de aprendizagem, fui alocado em 2 diferentes projetos: SmartStations, onde desenvolvi a página de *login*, recuperação de palavra-passe e cadastro; e TEM Marketing, no qual modernizei o visual do site para estar de acordo com o projetado pela equipa de design UI/UX.

Índice

Índice de Figuras

1.Introdução

Este relatório apresenta os resultados do estágio curricular do curso de Desenvolvimento de Aplicações Informáticas do Instituto Politécnico da Guarda, realizado entre 07 de Março de 2022 até 22 de Julho do mesmo ano.

Durante o curso de Desenvolvimento de Aplicações Informáticas iniciado no ano de 2020, adquiri conhecimentos nas diversas linguagens de programação, estudei conceitos e pratiquei os vários campos de desenvolvimento de *software*. Com esta experiência básica, pude perceber que gostei mais da parte de *front-end* das aplicações, embora tenha conseguido notas boas em todas as disciplinas. Esta preferência talvez se dá pela experiência prévia com design e pelo interesse geral pelo visual.

Com isto, procurei um estágio que pudesse aprimorar minha experiência e saberes com *front-end*, e consegui entrar em contacto com a empresa Shore Spun Lda com a ajuda do colega Ilberino, que aceitou meu pedido de estágio, focando em *front-end* e em *web design*. O plano inicial foi trabalhar com *mobile*, mas no decurso do estágio migrei para *desktop*. A empresa situa-se em Lisboa, e realizei o estágio de forma remota.

Utilizei meu próprio portátil para codificar, e os programas utilizados foram Visual Studio Code para escrever código, o browser FireFox para ver o resultado e inspecionar o código através do console, a *framework* Angular e NodeJS.

O objetivo do estágio foi aprender e aprimorar conhecimentos já adquiridos na graduação. No meu caso, aprendi um *framework* completamente novo, e desta forma também foi feito o primeiro contacto com a linguagem de programação do Angular, o *TypeScript*. Como Angular é um *framework* bem atual, foi extremamente importante dar os primeiros passos durante o estágio, no aprendizado tanto dos pormenores quanto a experiência de estar a trabalhar dentro de uma empresa em um projeto real.

Este relatório divide-se em 6 capítulos, sendo o quarto capítulo dividido em 6 subcapítulos, um para cada projeto realizado no decorrer do estágio. O relatório é finalizado com a conclusão, e as referências bibliográficas.

2. A Empresa

O Estágio foi realizado na empresa portuguesa Shore Spun Lda, situada em Lisboa desde 2015, que pertence ao supervisor do estágio, André Cabeças, formado em Gestão no Instituto Universitário de Lisboa (ISCTE).

A Shore realiza consultoria em tecnologia da informação para clientes em Portugal, Espanha, Estados Unidos e Inglaterra, conta com cerca de 130 colaboradores, entre developers, designers, recursos humanos, *helpdesk*, administração de sistemas, Cloud e arquitetura, e tem experiência e competências em sectores como a Banca, Seguros, Instituições Publicas, Telecomunicações e Consultadoria de Gestão (1).

Conforme se verifica do site da empresa, em 2019 tinha dentre seus colaboradores especialistas em (2):

Desenvolvimento aplicacional – .NET, JAVA, COBOL, PL/SQL, MOBILE;

Administração de Sistemas e Base de Dados;

Business Intelligence e RPA;

Análise Funcional e Técnica;

Gestão de projecto.

Dentro deste âmbito, meu plano de estudos focou na parte *front-end* de *Web Development*, com Angular, CSS e HTML como principais ferramentas a serem aprendidas e utilizadas.

3.Primeiros passos

No começo do estágio, realizado de forma remota, houve uma conversa com o meu supervisor André Cabeças e o *Developer* sênior que iria me auxiliar, Meigston Ramos, sendo acordado que na primeira fase do mesmo, eu deveria focar em aprender as ferramentas necessárias para entrar em um projeto posteriormente. Como não tive contacto com Angular durante a graduação, já que utilizamos ASP.NET na disciplina de Programação para Internet, me foi dado um link com 163 vídeos sobre Angular para aprender a utilizar essa plataforma de aplicações web (3).

Antes do começo do estágio eu havia pesquisado para aprender sobre AngularJS (porque achei que ia ser este Angular que ia utilizar durante o estágio), mas na vídeo aula, a professora explica que o AngularJS é na verdade o Angular 1, ou seja, é muito antigo e desatualizado.

Nos vídeos de aprendizado, foi uma batalha para conseguir deixar tudo a funcionar porque os vídeos são de 2017 e usam Angular 4. No momento do estágio, estamos no Angular 13, além de problemas de versão do NodeJS incompatíveis com meu Windows 7. Tive de buscar na Internet qual a versão compatível (13.14), e após instalar o AngularCLI (*Command Line Interface*, uma interface que permite executar operações de sistema dentro do Visual Code) e o NodeJS, para iniciar um projeto, deu erro de versão Node. Tive de desinstalar e buscar uma versão que funcionasse tanto para Windows 7 quanto para o AngularCLI (versão 12), ficando assim o Node funcional.

Nas primeiras semanas a atividade foi estudar os vídeos. Aprendi e percebi os conceitos, mas algumas coisas eram um tanto complexas, e possivelmente só seriam realmente aprendidas com um projeto real. Em certo momento, o supervisor foi informado do meu progresso até então, e ele enfático em dizer que eu deveria tomar o tempo que for necessário para assimilar os conteúdos.

Alguns conteúdos dos vídeos, como formulários, validações e redireccionamentos, havíamos aprendidos já na disciplina de Programação para Web do semestre passado, mas no Angular é diferente no sentido de que a construção dos sites é por módulos e componentes.

Com a base da disciplina supra mencionada, ao utilizar o Angular, foi às vezes até mais fácil implementar e perceber certas funcionalidades que são genéricas de todas as Web *Frameworks*. Isto ocorre porque o Angular possui uma biblioteca de componentes de interface (4) chamado Angular Material (5), que agiliza o desenvolvimento por possuir componentes como elementos de formulário, componentes de layout, botões, estilos de navegação, tabelas e modais já funcionais e editáveis. Assim, é possível personalizar tais componentes e colocá-los em funcionamento de forma mais fácil do que quando se utiliza ASP.NET.

Ao codificar junto com a apresentadora dos vídeos, pela razão da *playlist* ser antiga e utilizar versões depreciadas do Angular, alguns códigos e expressões não estavam mais a funcionar, o que dificultou bastante seguir de forma consistente o que estava a ser apresentado. Por diversas vezes, mas nem sempre, haviam comentários abaixo explicando o que deveria ser mudado, mas na maioria, tive de simplesmente parar de codificar junto pois do contrário não acabaria nunca de ver a playlist, que era extensa. Contudo, foi possível aprender bem as possibilidades e o *workflow* de construir um site com Angular.

Ao terminar de ver os vídeos, o *developer* de Angular da empresa me passou um pequeno projeto para, justamente, aprender, testar e praticar os conteúdos aprendidos.

4. Projetos

4.1 Pokedex

O projeto sugerido era a criação de uma "*Pokedex*": uma espécie de base de dados com todos os *Pokémons*. *Pokémon* é uma franquia de jogos e desenhos animados onde as personagens colecionam, capturam e batalham com criaturas fictícias. No jogo, uma *Pokedex* é utilizada como uma agenda que mostra ao jogador determinado Pokémon, com dados relativos a ele (tipo, altura, peso, entre outros). Ao pesquisar na Internet, há imensos vídeos e conteúdos sobre como desenvolver uma "*Pokedex*" utilizando diversas linguagens de programação, inclusive Angular. O *developer* passou-me alguma documentação e vídeos para basear-me, e propôs alguns extras que deveria colocar no projeto, como a opção de clicar em uma linha da tabela e esta abrir um *card* com mais informações – de forma que eu tinha de criar algo único para o meu projeto que não estivesse pronto na internet. A proposta era a criação de uma página única com Angular, que mostre *Pokémons* em forma de tabela, com uso de uma "*Application Programming Interface*" (API) - uma interface de comunicação que liga as diversas funções em um site de maneira que possam ser utilizadas em outras aplicações (6) - e que, ao clicar em um elemento, um novo *card* abrisse com mais informações.

Ao seguir os *links* enviados, haviam alguns vídeos interessantes como recomendados, então comecei a codificar seguindo outros tutoriais, ficando mais fácil o aprendizado com os vídeos de um outro *youtuber*. Estes vídeos vieram a somar com os conteúdos aprendidos nas semanas anteriores, e foi mais fácil de perceber certos conceitos em conjunto com uma prática mais focada em realizar o projeto.

No entanto, não estava a funcionar (e não sei como que para o *youtuber* funcionou) fazer a API mostrar toda a lista de *Pokémons* (ao invés de manualmente ter de adicionar cada um). Dava um erro quando tentava apresentar o nome do *Pokémon* no *card*, conforme é visto na Figura 1:

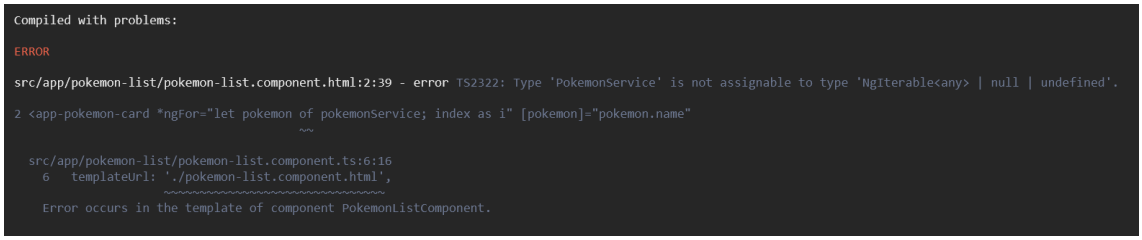


Figura 1: Erro de compilação

Para resolver este problema, fui ao vídeo de exemplo que o *developer* da empresa havia indicado. No vídeo, o apresentador constrói a página de forma diferente, ele começa pelo código e depois vai à interface (o vídeo que utilizei começou pela interface). Com nomenclaturas diferentes, consegui mesmo assim achar o que estava a faltar: nos vídeos de treino da playlist, a *youtuber* falava sempre da necessidade da expressão ".subscribe()" para fazer requisições HTTP. Como este outro vídeo falava disso, adicionei ao meu código.

```
15  ngOnInit(): void {
16    this.pokemonService.carregarPokemons()
17    .subscribe((response: any) => {
18      response.results.forEach((result: { name: string; }) => {
19        this.pokemonService.getMoreData(result.name)
20        .subscribe((uniqResponse: any) => {
21          this.pokemons.push(uniqResponse);
22        });
23      })
24    });
25  }
26
```

Figura 2: Resolvendo o nome dos *Pokémons*

Teve-se de mudar algumas coisas que estavam a dar erro, possivelmente pelas versões diferentes do Angular, além de que os nomes dos *Pokémons* não estavam a aparecer. Como é possível ver na Figura 2, na linha 19, há a adição do result.name, que era justamente o "nome" que o Angular não estava a achar. Para chegar a essa conclusão, percebeu-se que os nomes estavam a funcionar quando não pegava nada da API – ou seja, ao adicionar cada nome de *Pokémon* manualmente, na forma como ele havia feito antes (ele demonstrou como fazer com e sem a API), era só adicionar os nomes dos *Pokémons* em uma "Array[]", que os *Pokémons* eram adicionados com suas imagens, mas o nome do *Pokémon* era o que fosse digitado no

Array, portanto poderia haver erro humano. Com a API todo esse processo é automático, e os nomes estão prontos e em ordem.

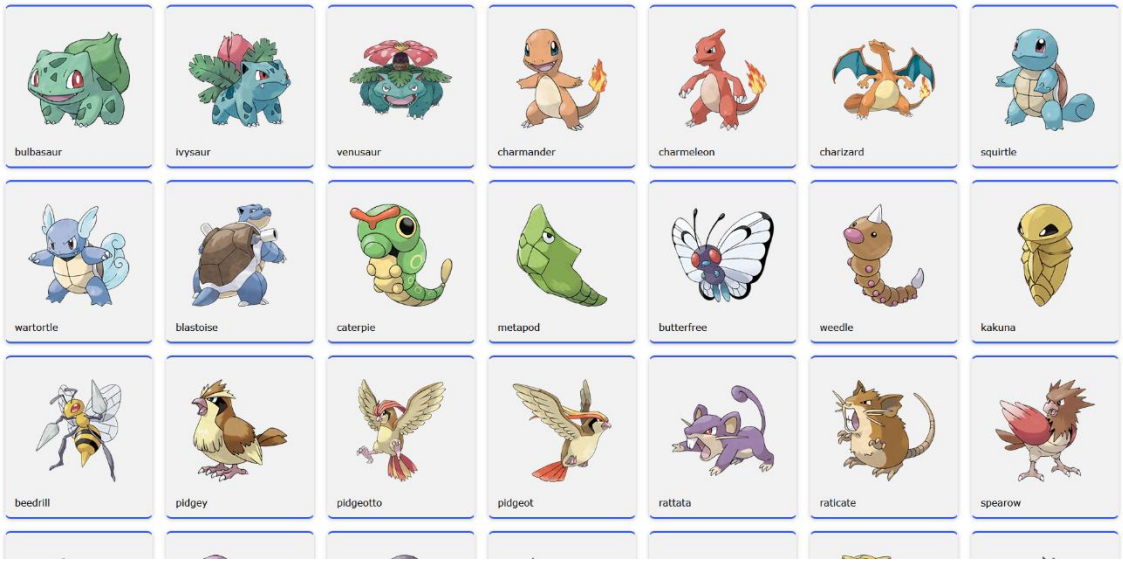


Figura 3: A primeira versão da *Pokédex*

Após ter feito o projeto seguindo um vídeo tutorial no *youtube*, percebeu-se que o *developer* da empresa pediu para que a página fosse feita em estilo de tabela, com linhas. Dessa forma, foi copiado o projeto e começado uma segunda versão, onde foi adicionada a paginação e o modelo básico de tabela com Angular Material que foi sugerido.

Um problema que surgiu neste momento foi que, com o tutorial utilizado, o *youtuber* fazia a tabela de forma que, toda vez que clicássemos em uma linha, uma nova página abria com os dados extras dos *Pokémons*. É isto que o *developer* da empresa pediu, mas que ocorresse sem o redireccionamento à uma nova página: o objetivo é que os dados extras apareçam em uma *popup*, chamada "*Dialog*" no Angular.

Aqui foi passado a maior parte do tempo para fazer funcionar. Em determinado momento, o *dialog* abria, mas os dados da API não estavam a passar. Depois de muito tempo, dias, chegou-se ao problema principal:

```
//AQUI ESTA O PROBLEMA, TEM QUE TENTAR FAZER ESSE ID ENTRAR NO GETPOKEMON()
getPokemon(id: any){
 // id = this.poketable.selectedRow;
 id = '2';
    this.pokemonService.getPokemons(id).subscribe(
      res => {
        this.pokemon = res;
        this.pokemonImg = this.pokemon.sprites.front_default;
        this.pokemonType = res.types[0].type.name;
      },
      err => {
        console.log(err);
      }
    )
}

onNoClick(): void{
  this.dialogRef.close();
}
```

Figura 4: ID do *Pokémon*

Ao dizer qual é o valor da "*ID*" (Figura 4), conseguia fazer a *dialog* abrir com os dados do *Pokémon* referente à *ID*, mas não que essa *ID* fosse relativa com as mesmas informações da linha clicada, conforme Figura 5 abaixo.

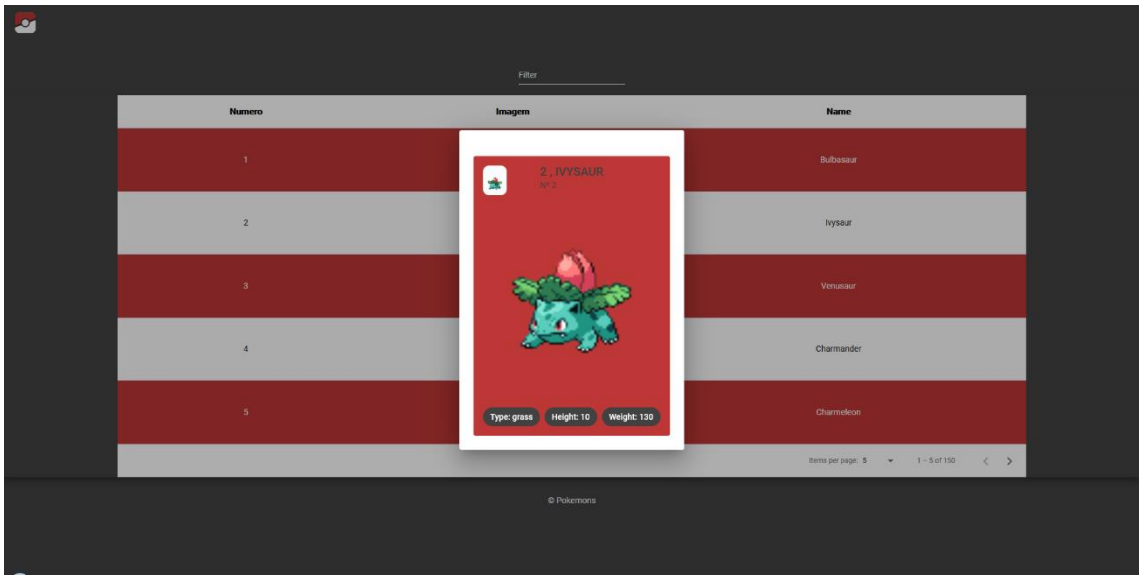


Figura 5: *Dialog* mostra *o Pokémon* mas não consome API

O problema é que ao informar o *ID* manualmente, é estático, ou seja, todas as linhas que se clique mostram o mesmo *Pokémon*. O correto é que a mesma API que mostra todos os itens na tabela identifique o *dialog* clicado e forneça os dados corretos e únicos.

Para resolver o problema, após pesquisar, pareceu correto utilizar uma forma de envio de dados entre dois componentes distintos. Foi visto um vídeo que explicava como era possível receber e enviar mensagens utilizando apenas um Serviço (que eu já estava a fazer no meu projeto, mas apenas do serviço para outro componente – o serviço é o que liga a API usando um protocolo HTTP), ou seja, o serviço é apenas um "terceiro" que faz a "ponte" entre os dois componentes. Assim sendo, foi colocado código no serviço (Figura 6), e agora tinha de descobrir onde colocar nos componentes. Para receber era fácil, como era sabido que o *ID* tinha de mudar dentro da função "getPokemon(id)".

Figura 6: O código no Serviço

Para enviar a mensagem, contudo, tentou-se colocar dentro da função de "getPokemons" do outro componente, mas não funcionou da forma como era esperada: pegava os dados apenas do último *Pokémon*, pois estava ao final do *loop For* (um pedaço de código que corre repetidamente até que uma certa condição tenha sido atendida). Assim, como antes de solucionar o problema já imaginava-se que a forma de conseguir esse ID era com o número da linha ("selectedRow"), porque cada linha tem 1 *Pokémon*, então o valor do *ID* é o mesmo que o valor da linha, e assim foi posto dentro da função "getRow()", como mostrado na Figura 7. Esta função chama o *Dialog*, define que a mensagem a ser enviada é o número da linha, e envia esta mensagem para o componente "filho":

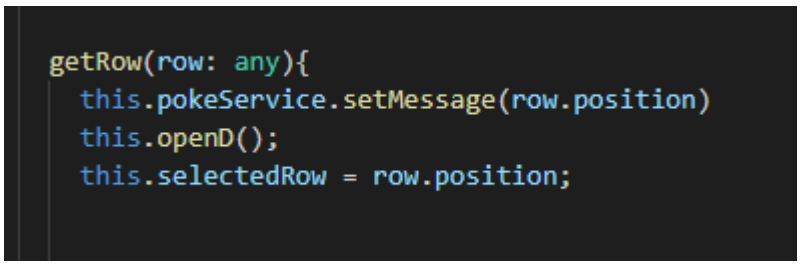


Figura 7: Definindo a mensagem a ser enviada

Para finalizar, foi necessário simplesmente dizer que o ID da função tem o mesmo valor da mensagem recebida, ou seja, do valor da linha a qual se clicou, neste caso “this.pokemonService.getMessage()” (Figura 8).

```
getPokemon(id: any){
    id = this.pokemonService.getMessage();
    this.pokemonService.getPokemons(id).subscribe(
      res => {
        this.pokemon = res;
        this.pokemonImg = this.pokemon.sprites.front_default;
        this.pokemonType = res.types[0].type.name;
      },
```

Figura 8: ID funcional

O resultado era o que se esperava, a cada linha da tabela clicada, o *Pokémon* referente aparece no *Dialog popup*, com as informações e imagens contidas na API (Figura 9).

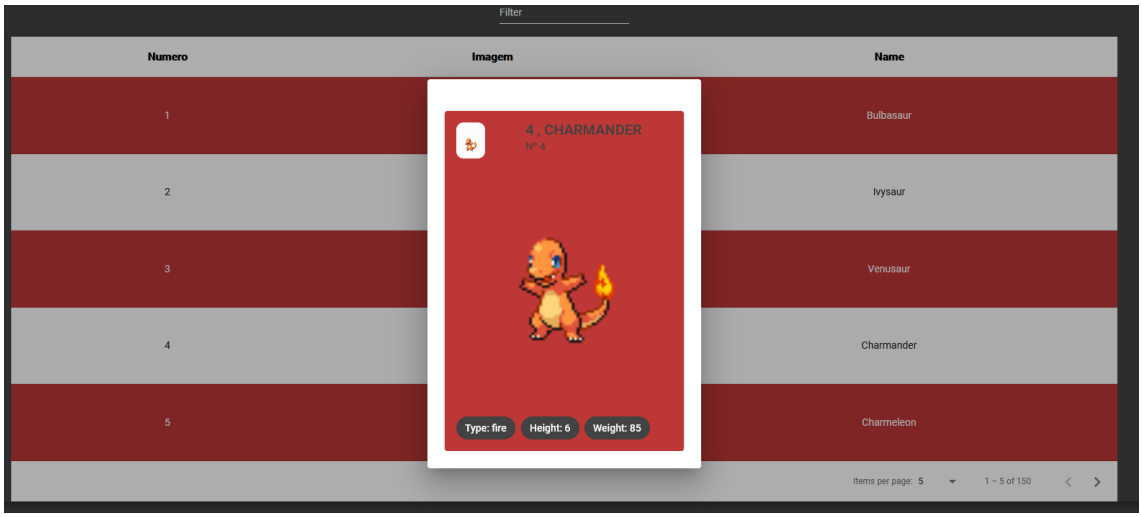


Figura 9: *Dialog* funcional

Este projeto foi muito importante por ser o primeiro contato direto e real com Angular em que foi produzido algum resultado mais profissional, além da necessidade de resolução de problemas que surgiram e não estavam resolvidos no tutorial que foi seguido.

4.2 Google Maps

A seguir, o *developer* me atribuiu a atividade de aprender a integrar a API do Google Maps em um projeto Angular.

Para isso, deu-me um link para criar uma chave de API pelo *Google Cloud Platform* (7). Com esta chave criada, integra-se ao código *Typescript* ou *HTML* de um projeto para ativar as funções de serviço do *Google Maps*. A partir daí, foi necessário implementar o que se pretendia, como deixar marcadores de local móveis com o rato, ou fazer aparecer mais de um marcador no mesmo mapa.

Quando conseguiu-se colocar tudo no projeto que foi criado de testes, utilizando o módulo "agm@core" que possibilita a utilização do *Google Maps* em componentes do Angular, ocorreu um erro de versões. Para resolver, foi tentado modificar as versões do "agm@core", mas sem sucesso.

O que resolveu foi fazer um *rollback* – voltar à versão anterior do *Google Maps*, como se vê na Figura 10, porque a nova versão do *Google Maps* havia modificado algumas nomenclaturas.

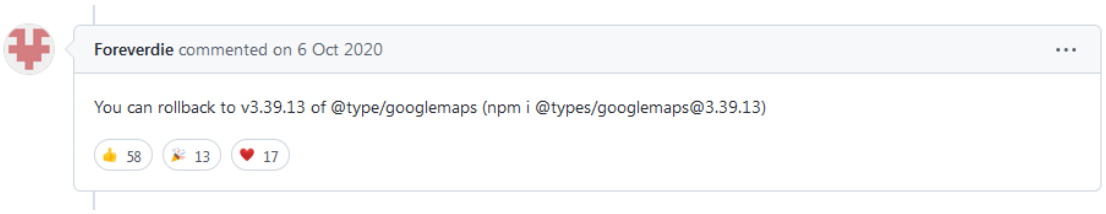


Figura 10: Rollback de versões

Da mesma forma, foi seguido um tutorial para colocar um marcador que diz a posição de latitude e longitude do mesmo em uma *label* abaixo do mapa. Só que a propriedade *MouseEvent* estava a dar erro nas propriedades das coordenadas. Após pesquisa, foi achado o erro de que, nesta versão do *Google Maps*, utiliza-se "$event: google.maps.MouseEvent" ao invés de apenas "@event: MouseEvent" na condição da função.

```
markerDragEnd($event: MouseEvent) {
  console.log($event);
  this.lat = $event.latLng.lat();
  this.lng = $event.latLng.lng();
  this.getAddress(this.lat, this.lng);
}
```

Figura 11: Erro de versão

Ao fazer a troca, as propriedades “latLng.lat” e “latLng.lng” passaram a funcionar, e consequentemente, o mapa. Toda a vez que se move o marcador, a latitude e longitude do local é mostrada na *Label*.

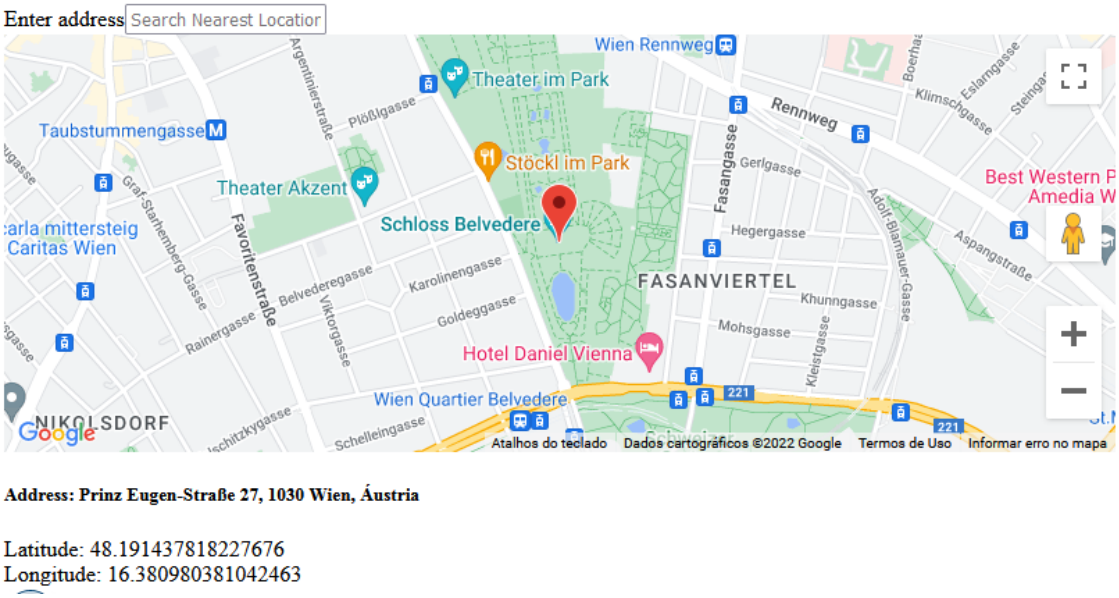


Figura 12: Latitude e Longitude em mapa do *Google Maps*

Posteriormente, o *developer* pediu para que fosse integrado algo com roteamento de páginas e praticar diferentes opções para os *markers* e *pins* do *Google Maps*, então criou-se um site com menu de navegação no topo, com uma página inicial que tem *links* de roteamento para outras páginas, e um mapa com 3 marcadores. No entanto, não foi possível implementar mais de um marcador no mapa que já mostrava Latitude e Longitude. Isto foi feito então com um novo mapa em outra sessão da página, conforme Figura 13.

Figura 13: Mais de um marcador no mesmo mapa

A página com a *navbar* no topo, de forma bem simples, pode ser vista na Figura 14:

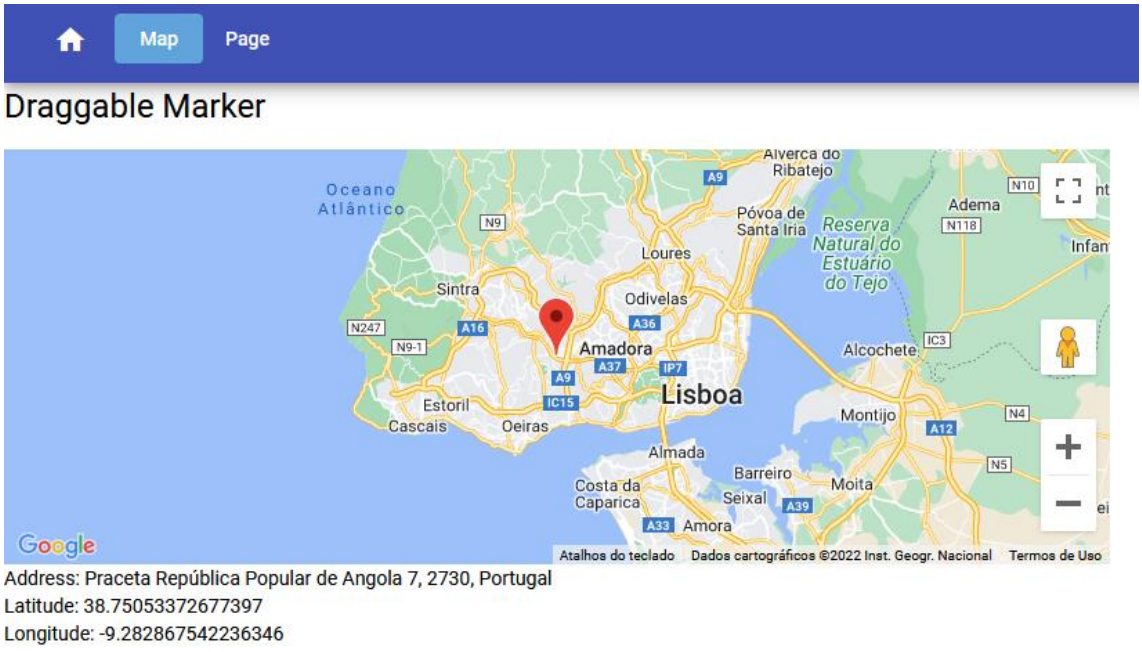


Figura 14: Marcador móvel

O botão "Map" mostra o mapa móvel que muda a latitude e longitude conforme o posicionamento do marcador. No segundo botão do menu ("Page"), mostra um mapa com 3 *markers* de localização pré-definidas. Aqui tentou-se fazer um layout um pouco mais bonito para as *InfoWindows* (janelas que abrem quando se clica no *marker*), que são nativas do Angular *Google Maps*.

Para praticar um pouco mais, pensou-se em fazer um tipo de página como se tem em supermercados, com as filiais a mostrar no mapa, ao clicar no *marker* da localização do supermercado no mapa, uma janela com as informações aparece (Figura 15).

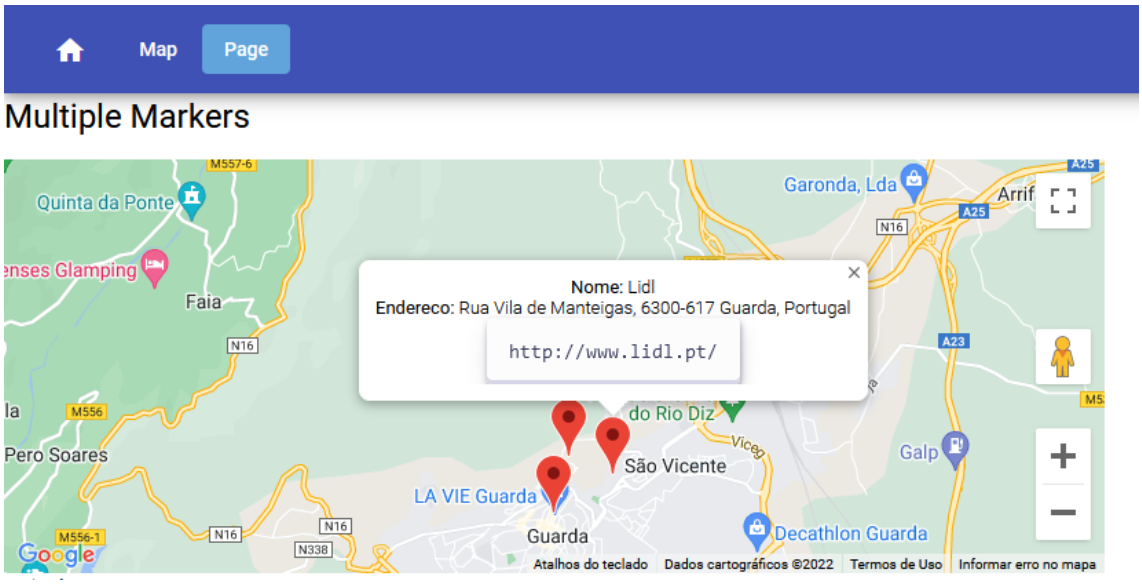


Figura 15: Janela com informações do *marker* clicado

Com o código mostrado na Figura 16 abaixo, conseguiu-se fazer com que, ao clicar em um outro *marker* ou no mapa, a *InfoWindow* que já estivesse aberta fechasse, assim somente uma *InfoWindow* é aberta por vez:

```
close_window(){
  if (this.previous_info_window != null ) {
    this.previous_info_window.close()
    }
  }

  select_marker(infoWindow: any){
    if (this.previous_info_window == null)
     this.previous_info_window = infoWindow;
    else{
     this.infoWindowOpened = infoWindow
     this.previous_info_window.close()
    }
    this.previous_info_window = infoWindow
  }
```

Figura 16: Apenas uma *InfoWindow* aberta por vez

4.3 HTML e CSS

Este projeto teve como objetivo aprender mais e praticar CSS e HTML de forma mais "pura", sem o auxílio do *Framework* Angular, mas cujos conhecimentos são aplicados de mesma forma. Foi utilizado um tutorial no *Youtube* com boas recomendações, e definitivamente aprendeu-se imenso, desde conceitos básicos, como o posicionamento dos elementos no ecrã com CSS, até a utilização mais profissional e meticulosa de divisão/secção (*Div*) do elemento HTML, a alteração do visual de botões, fontes, títulos (conforme Figura 17) e a criação de menus e formulários sem o auxílio do Angular Material utilizado nos projetos anteriores.

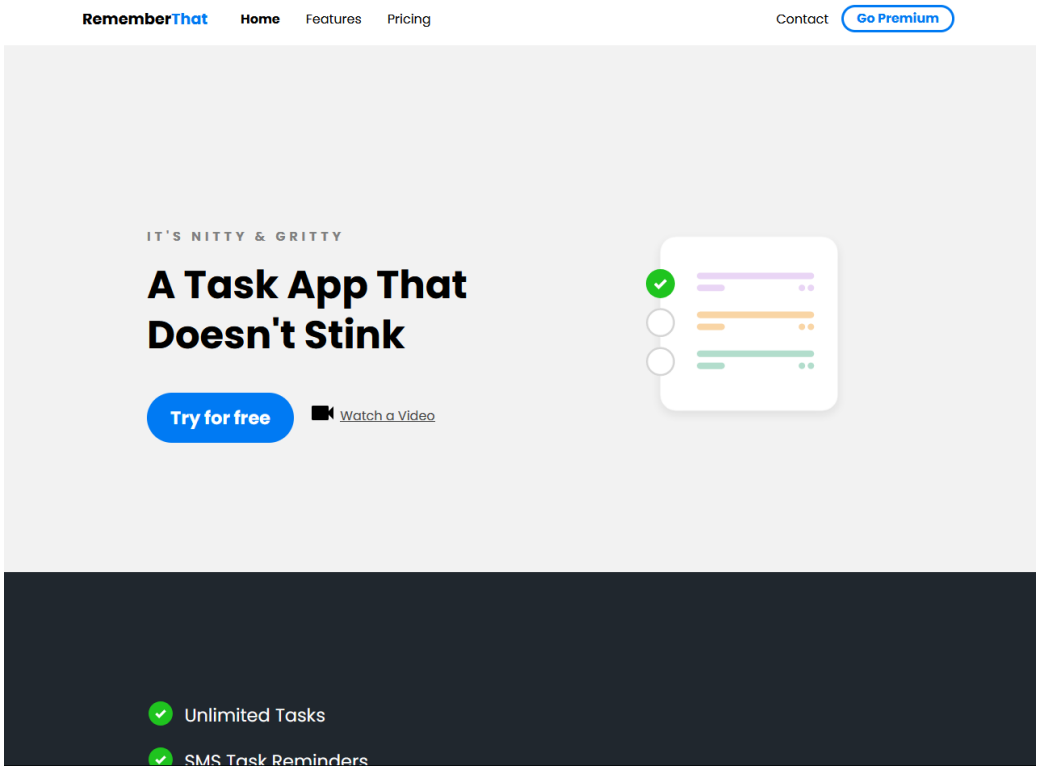


Figura 17: *Layout* da página concluída

Além disto, o projeto também utilizou uma forma mais básica do Google Maps (Figura 18), sem o AGM@Core do Angular, e pode-se perceber que nesta forma, há menos poder de edição e funcionalidades. Neste sentido, observa-se o quanto o módulo do Angular auxilia os desenvolvedores e adicionar elementos úteis de navegação.

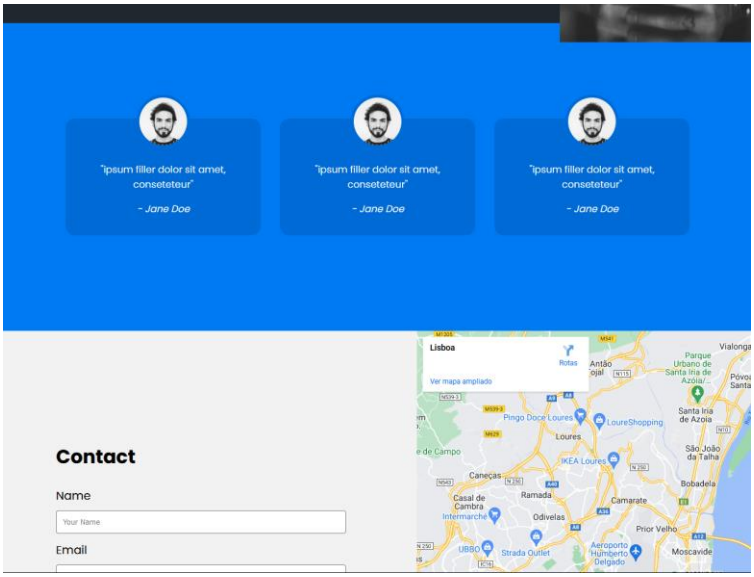


Figura 18: Integração *Google Maps* e formulário

O projeto também ensinava a deixar a página responsiva, e o maior elemento disto é o menu "sanduíche" para telemóveis (Figura 19). Com a utilização do elemento "@media" no CSS, é possível definir o tamanho do ecrã que terá os elementos em determinado tamanho e se os elementos mudam de posição ou tamanho nas variações de ecrã, inclusive podendo esconder elementos, como é o caso, onde o menu sanduíche clicável some se o ecrã é grande e os botões e links da *Navbar* aparecem no topo.

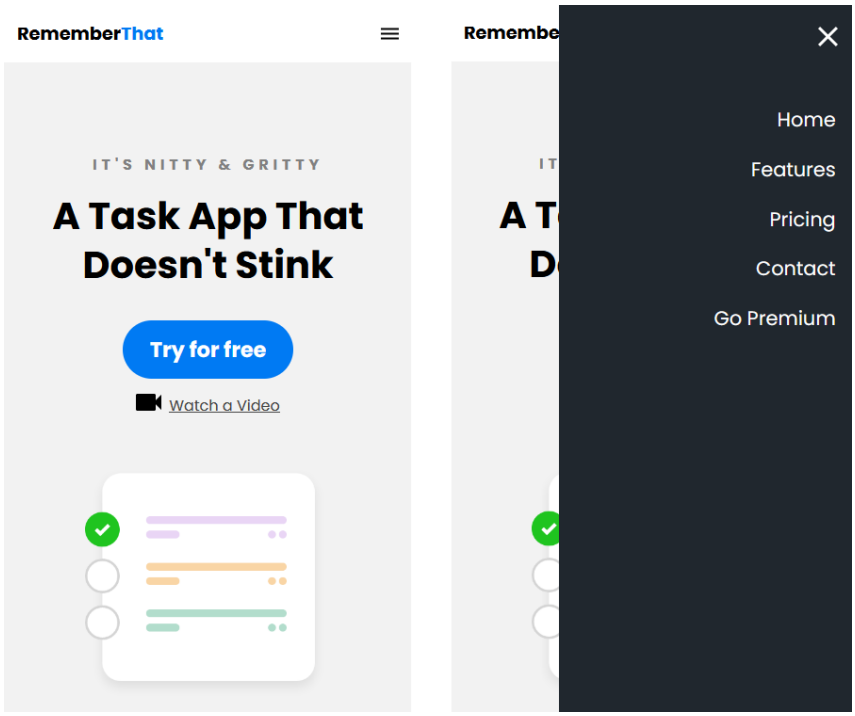


Figura 19: Menu "sanduíche" para telemóveis

4.4 Smart Stations: login

O desenvolvedor sênior da empresa me contactou para inserção em um projeto que a empresa está a trabalhar. Na chamada, além da explicação do projeto, foram tiradas dúvidas acerca da vida de desenvolvedor, como trabalhar em equipa, como desenvolver com mais produtividade com Angular e organizar os ficheiros dos projetos.

O projeto que a empresa está a desenvolver se chama Smart Stations, e é basicamente uma espécie de aplicação para entrega de bens concentrada em um *hub* físico. É bem similar à empresa *iFood* que temos no Brasil. O utilizador vai à aplicação, tem um login, uma lista de restaurantes e itens que lhe é oferecido, e pode finalizar o pedido, sendo então um entregador atribuído para sua encomenda.

O *developer* mostrou um pouco da organização do código do projeto no Angular, com os componentes e módulos, e os ecrãs feitos pelo *designer* UI/UX, no *Figma*, o qual ele utilizava para desenvolver seus componentes e organizar tudo.

Cada elemento do site foi feito separado, visto que, de acordo como explicado pelo *developer*, no Angular o que fazemos é "montar um quebra-cabeças", primeiro o *developer* fez cada elemento separado como componente no Angular, e depois disso, ele foi aos poucos juntando os componentes para gerar uma página, desta forma elementos que se repetem podem facilmente ser inseridos e modificados individualmente, a depender da necessidade.

A tarefa que me foi dada foi de fazer o ecrã de login da aplicação, seguindo o *design* feito no Figma, que pode ser visto na Figura 20:

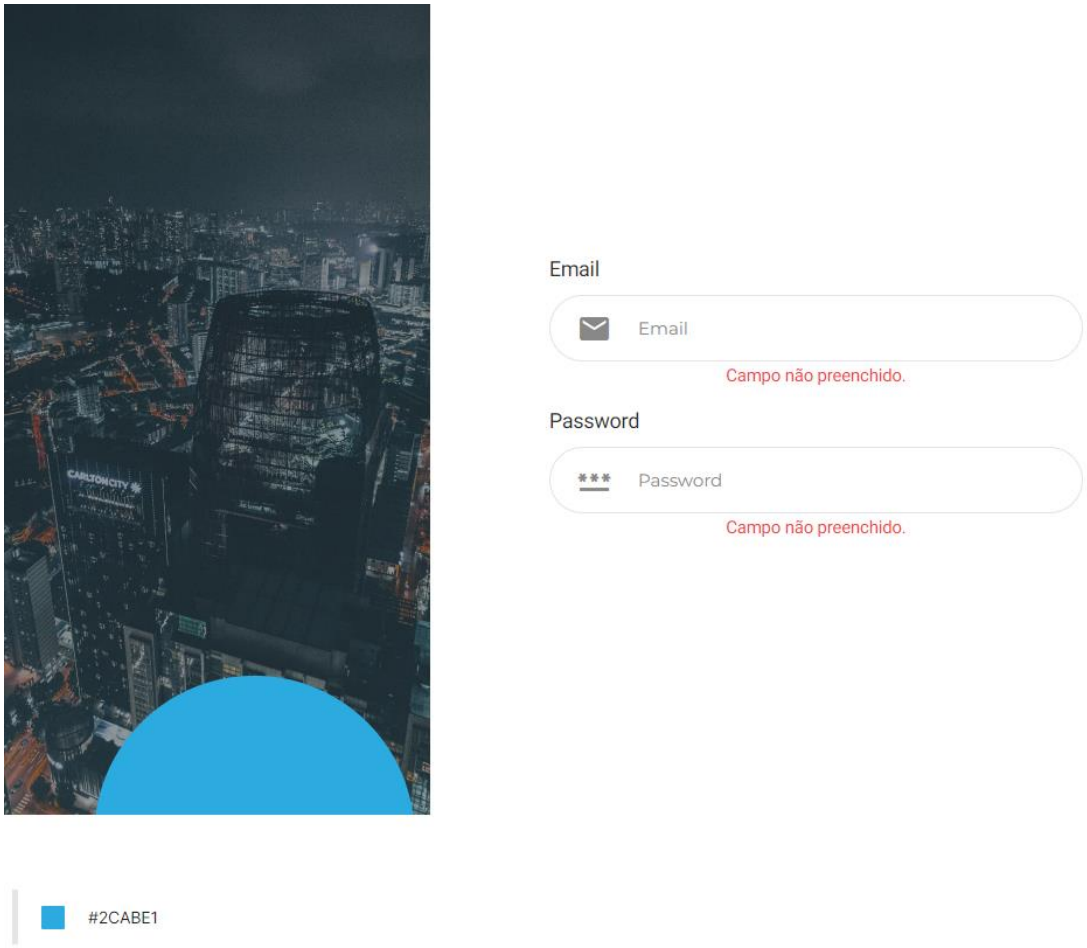


Figura 20: Design do ecrã de login

Aqui o projeto anterior de HTML e CSS teve fundamental importância. Comecei por colocar todos os elementos HTML e depois posicionar com CSS. Houve alguma dificuldade aqui principalmente em relação ao tamanho da imagem, e várias tentativas até conseguir posicionar corretamente os elementos.

Um primeiro desafio foi inserir os ícones do email e da palavra-passe dentro do campo de *input*. Como foi utilizada a propriedade do Angular Material para criar o elemento *input*, a forma de inserir o ícone é diferente, portanto precisou-se uma pesquisa, várias tentativas de opções que já estavam obsoletas, até conseguir chegar ao resultado ideal.

Para modificar o campo *input* do email e *password* de forma que ficassem arredondados, não bastou apenas utilizar a propriedade "*border-radius*" do CSS, porque como é um elemento do Angular Material, é necessário alterar o código-fonte do elemento, ou seja, com um comando diferenciado e próprio, modifica-se o próprio código do Angular Material. Para isto, precisou primeiro achar qual a sequência de código que altera os campos de *input*

(.mat-form-field), e seus sufixos (appearance-outline, outline-end e outline-start), e utiliza-se o comando "::ng-deep" para sinalizar ao Angular fazer a modificação em sua fonte:

```
::ng-deep .mat-form-field-appearance-outline .mat-form-field-outline-end {
    border-radius: 0 28px 28px 0 !important;
}

::ng-deep .mat-form-field-appearance-outline .mat-form-field-outline-start {
    border-radius: 28px 0 0 28px !important;
    min-width: 28px !important;
}
```

Figura 21: Alterando elemento do Angular Material

O elemento "!important" avisa o Angular que está a acontecer um *override* (sobreposição) do código fonte. Sem isto, a alteração não funciona.

Outra dificuldade foi colocar a bola azul na foto. Fez-se uma bola azul com *Photoshop*, e inseriu-se no HTML, e com o comando "*overflow: hidden*" em uma *div* própria, a bola parece cortada ao pé da imagem.

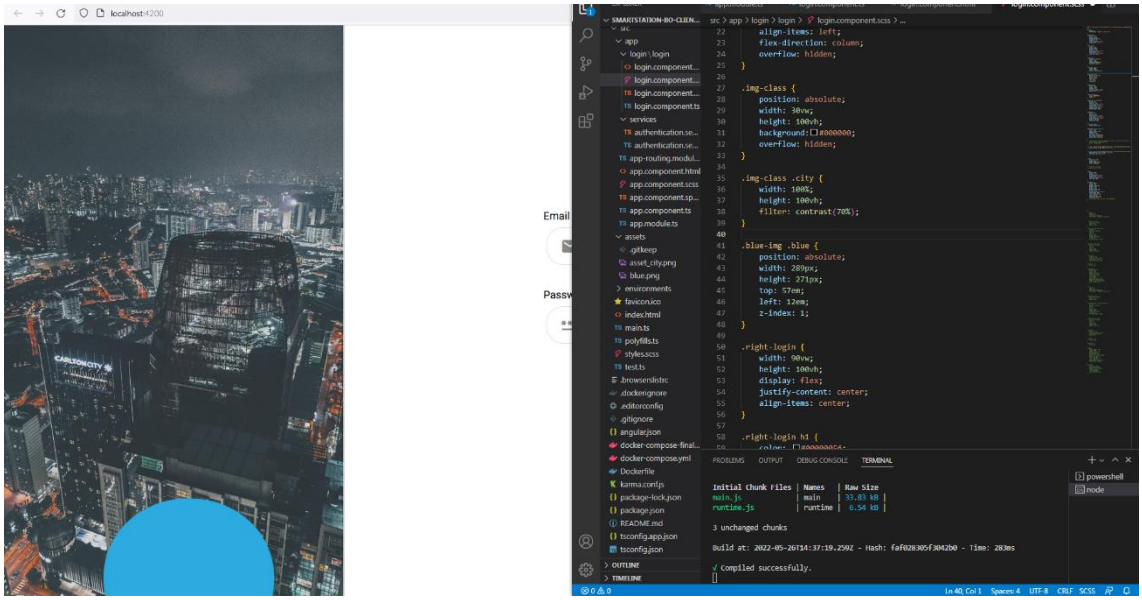


Figura 22: Imagem do ecrã de login com código CSS utilizado

Depois disto seguiram-se as validações dos campos de *input*, conforme pode ser visto na Figura 23:

```
loginForm!: FormGroup;

constructor(
  private authService: AuthenticationService,
  private router: Router
) { }

ngOnInit(): void {
  this.loginForm = new FormGroup({
    email: new FormControl(null, [
      Validators.required,
      Validators.email,
      Validators.minLength(6)
    ]),
    password: new FormControl(null, [
      Validators.required,
      Validators.minLength(3)
    ])
  })
}

onSubmit(){
  if(this.loginForm.invalid){
    return;
  }
  this.authService.login(this.loginForm.value)
  .subscribe(
    res => console.log(res),
    err => console.log(err)
  );
}
```

Figura 23: Validações dos campos em *Typescript*

O complemento das validações do *Typescript* ocorre no HTML, conforme Figura 24, o próprio Angular tem elementos de validação de campos, o elemento "<mat-error>". No momento em que a validação descrita no *Typescript* ocorre, o HTML mostra o que foi escrito entre as *tags* ("Campo não preenchido").

```html
<div class="right-login">
    <div class="card-login">
        <h1>Login Page</h1>
        <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
            <div class="textfield">
                <mat-label>Email</mat-label>
                <mat-form-field appearance="outline">
                <input matInput placeholder="Email" type="email" formControlName="email">
                <mat-icon class="icon" matPrefix>email</mat-icon>
                <mat-error *ngIf="loginForm.controls.email.errors">Campo não preenchido.</mat-error>
            </mat-form-field>
            </div>
                <div class="textfield">
                <mat-label>Password</mat-label>
                <mat-form-field appearance="outline">
                <input matInput placeholder="Password" type="password" formControlName="password">
                <mat-icon class="icon" matPrefix>password</mat-icon>
                <mat-error *ngIf="loginForm.controls.password.errors">Campo não preenchido.</mat-error>
            </mat-form-field>
            </div>
            <div class="button">
            <button class="btn-login" mat-flat-button color="primary" type="submit">Login</button>
            </div>
            </form>
        </div>

    </div>
</div>
```

Figura 24: Validação de campos em HTML

O resultado final das validações, no ecrã, é visto na Figura 25:

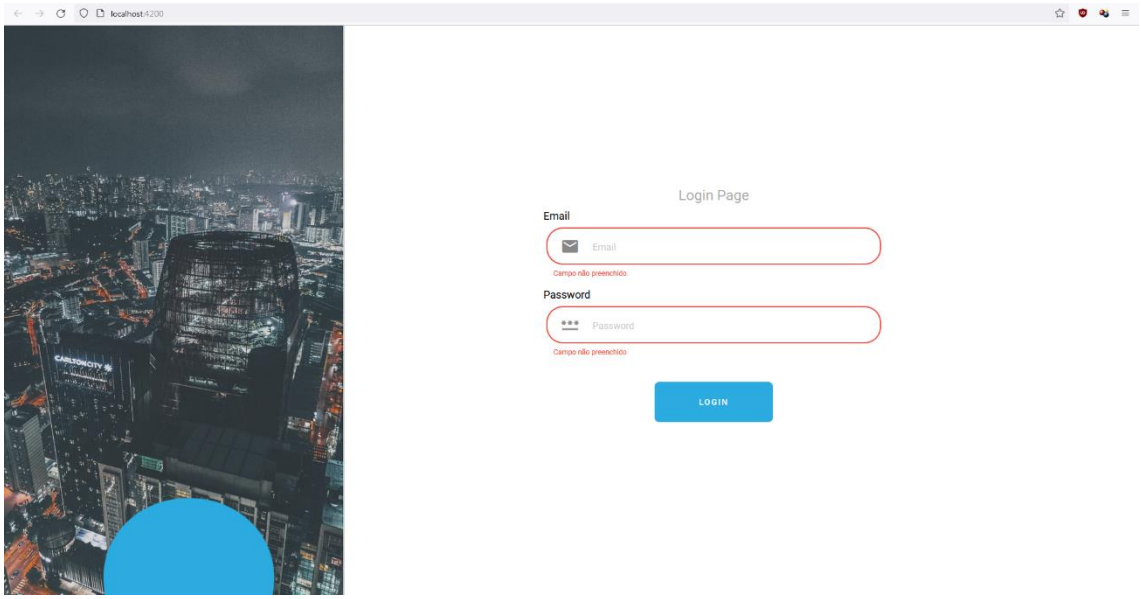


Figura 25: Validações de campos no ecrã

Na sequência, foi pedido pelo *developer* da empresa para deixar a página de login responsiva. Como sempre se deve começar como "*mobile first*", foi levado mais tempo do que levaria se tivesse começado a organizar o design pensando em telemóveis primeiro.

Houve necessidade de fazer engenharia reversa, ou reescrever o código CSS. Para auxílio como estou no começo da profissão, foi posto cor nas diversas *Divs* para melhor orientação graficamente no site (Figura 26), e foi-se aos poucos alterando valores para conseguir chegar em uma posição mais adequada dos campos de *input*, botões e textos quando o ecrã está em resolução menor.

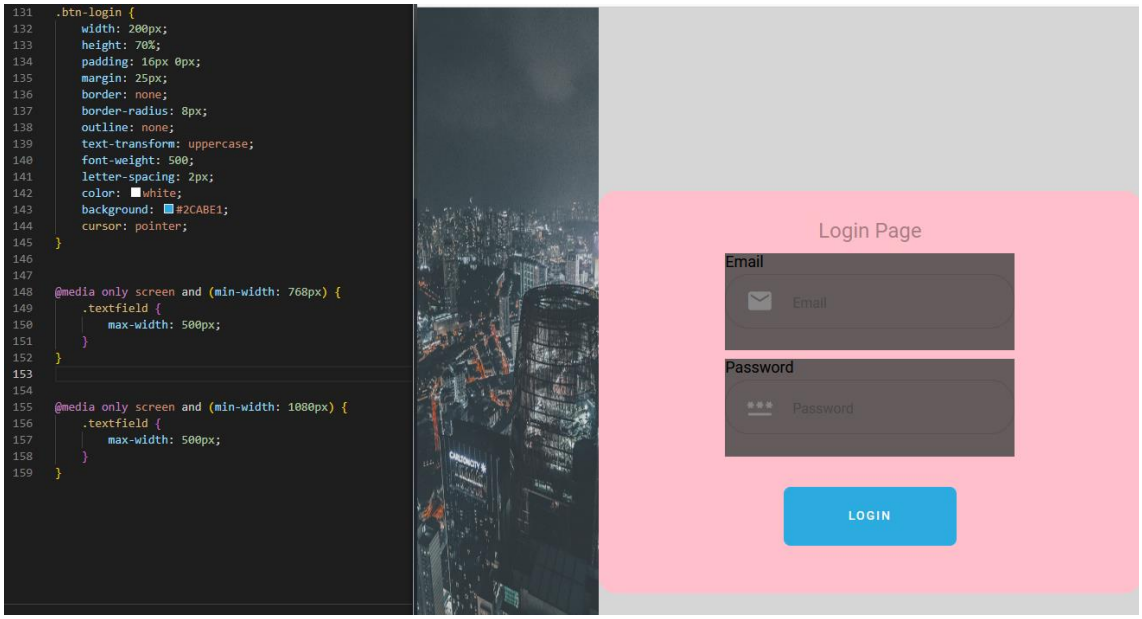


Figura 26: Organização de *divs*

Após isto foi pedido para que se adicionasse mais as páginas de cadastro (Figura 27) e de recuperação de palavra-passe (Figura 28):

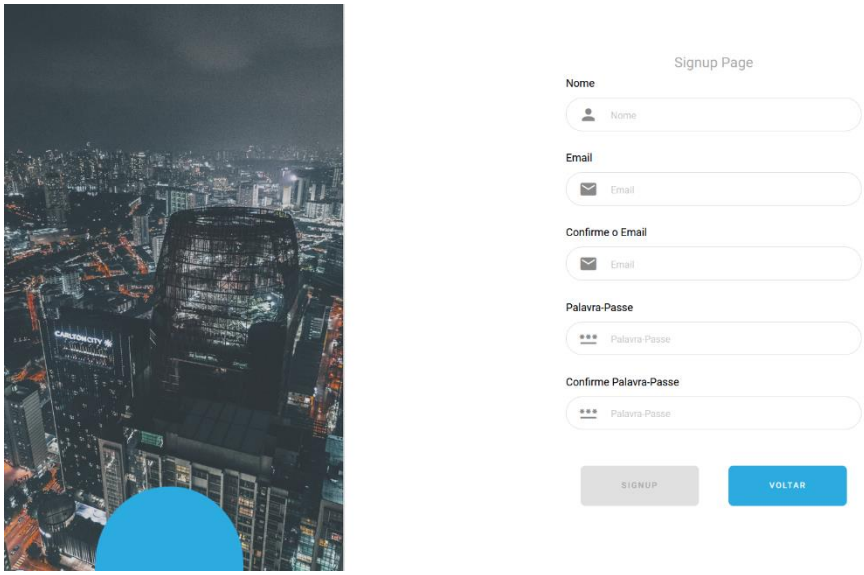


Figura 27: Cadastro

Além disto, enquanto todos os campos não estão preenchidos, o botão que envia os dados fica desativado, sendo graficamente representado pela cor cinzenta.

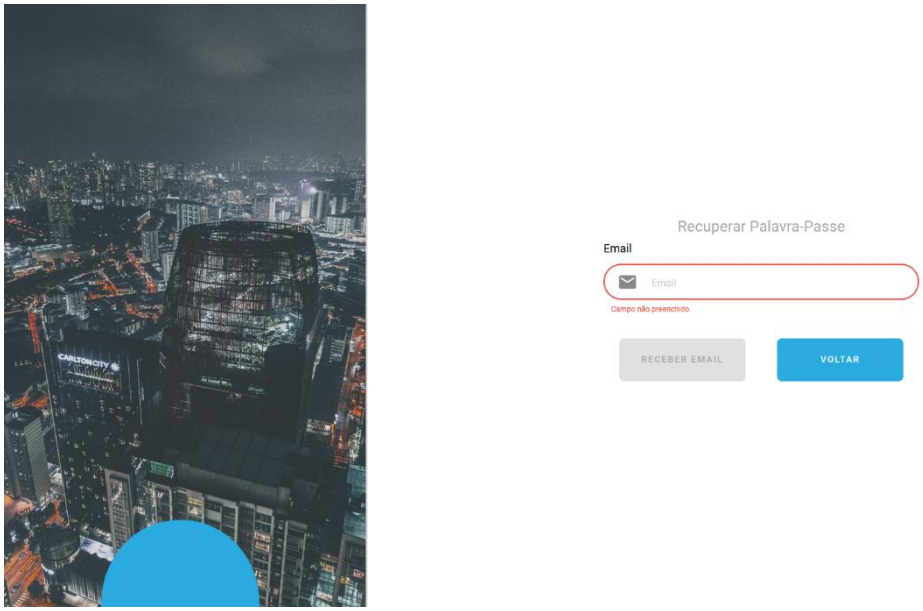


Figura 28: Recuperação de palavra-passe

Com esta adição, a página de Login ganhou 2 novos botões, sendo que o de recuperação de palavra-passe é mais discreto (Figura 29):

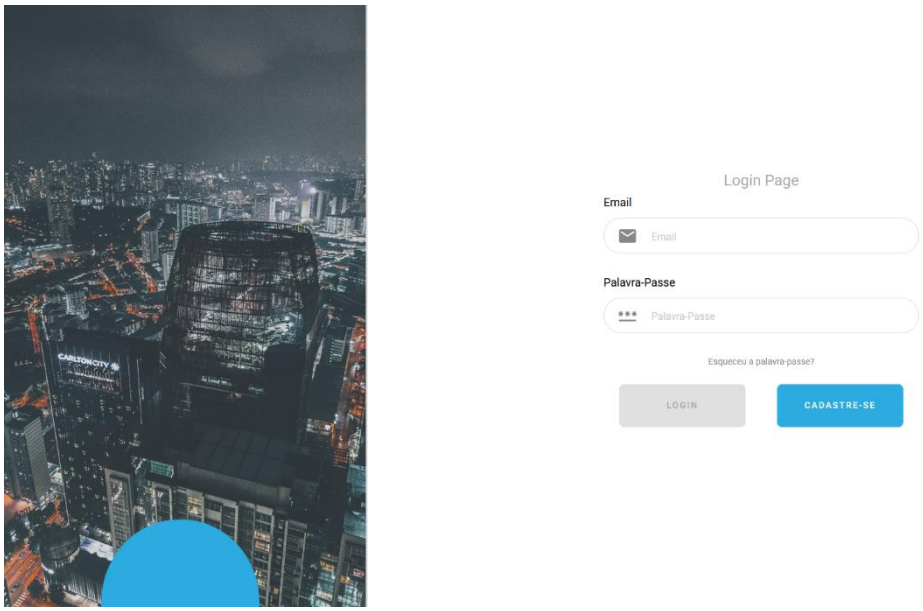


Figura 29: Ecrã de login

E o mesmo ecrã de login é mostrado na Figura 30 na sua versão de telemóvel:

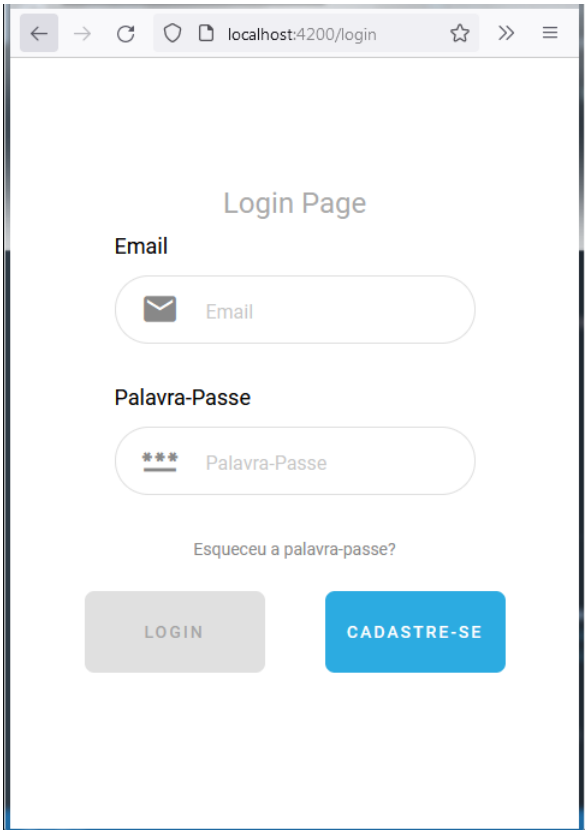


Figura 30: Ecrã de login no telemóvel

4.5 Catálogo de Jogos

Este foi um projeto próprio achado no *Youtube* para praticar a criação de sites com elementos Angular e que consumisse uma API para pegar todas as informações e imagens de jogos. Contém filtros, busca, e é basicamente um catálogo de jogos (Figura 31) parecido com o projeto de *Pokedex* criado anteriormente, onde a API mostra toda a informação em forma de tabela além de ter ícones das plataformas em formato *Scalable Vector Graphics* (SVG - uma forma de imagem em vetor que é possível de ser alterada através de código XML).

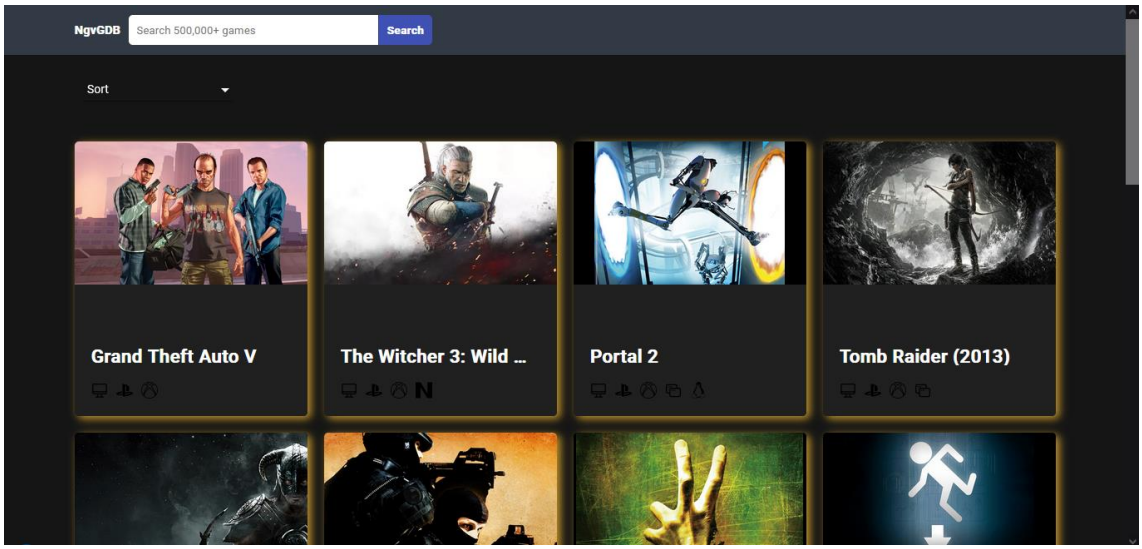


Figura 31: Catálogo de jogos

Foi um bom exercício não só para praticar o código *TypeScript* como também HTML e CSS, embora o foco do projeto estava mais na API e outras funções do site para filtros e buscas.

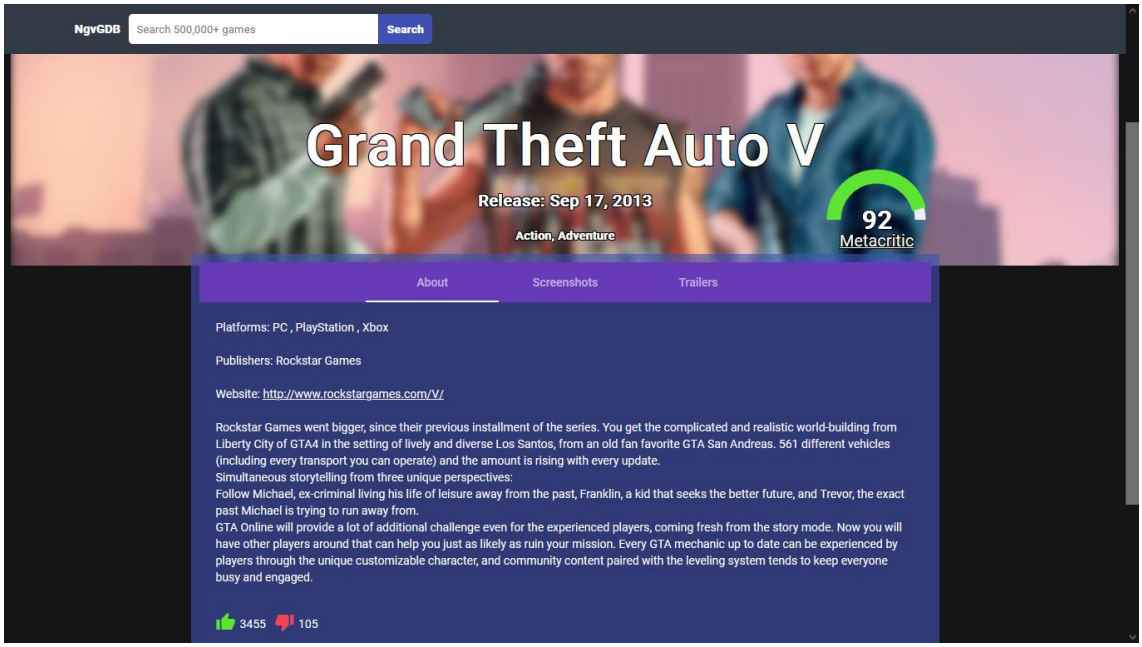


Figura 32: Página de jogo

Ao clicar no *card* de algum jogo, o utilizador é redirecionado para a página do mesmo (Figura 32), com mais informações que também são disponibilizadas através de API. A página de cada jogo contém uma *navbar* com informações, imagens e trailers, além da pontuação do site Metacritic, que é animada graças à um componente do Angular.

4.6 TEM – Tudo em Marketing

Fui alocado ao projeto TEM Marketing – Tudo em Marketing, com o objetivo de desenvolver o *redesign* do *site*, que foi feito pelo time UI/UX da empresa. Assim, precisou-se verificar o visual do site no momento (Figura 33):

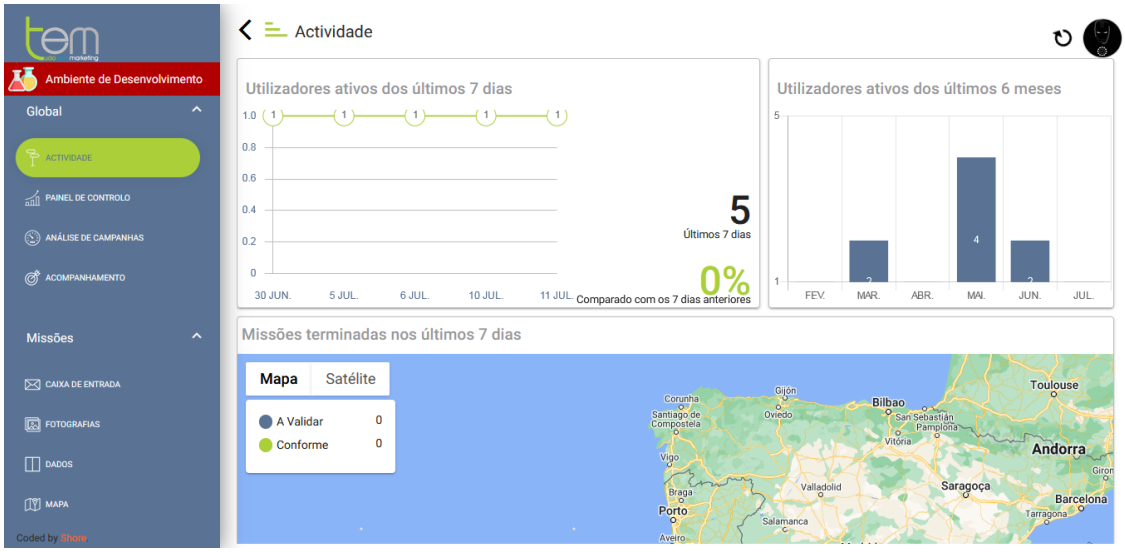


Figura 33: TEM Marketing antigo

Com acesso aos ficheiros feitos pelos *designers* em Adobe XD, que seriam utilizados para fazer as modificações necessárias a ponto de deixar o *site* funcional com a mesma aparência e *design* do protótipo (Figura 34):

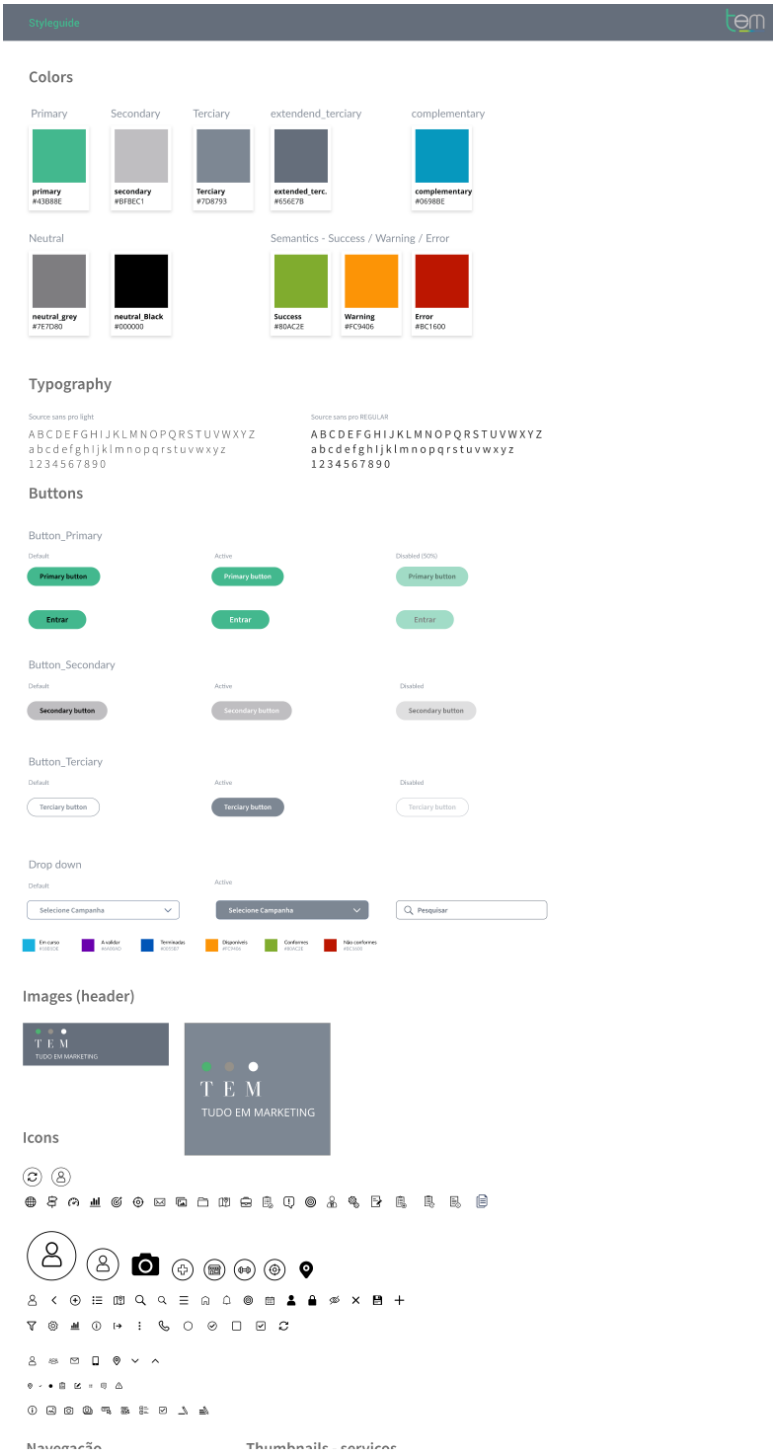


Figura 34: Alguns elementos que foram disponibilizados pelos designers

Para iniciar o projeto, o *Developer* sênior auxiliou com dicas de como abrir o projeto pelo portátil que estava a utilizar, inclusive com questões mais burocráticas como definir contador de tempo de trabalho na plataforma Slack e instalar Git para fazer *commit* (um tipo de ponto de salvamento para o qual um desenvolvedor consegue voltar caso ocorram alterações no código ou comportamento que precisem ser desfeitas no futuro) e *push* (o envio

do ponto salvo com o *commit*, do repositório local para o repositório remoto que normalmente fica no GitHub) diretamente pelo Visual Sudio Code para dentro do projeto ao final de cada dia, os quais ele iria revisar e aceitar.

Começou-se então a familiarização com o código já produzido pelo *Developer*, visto que o projeto todo já estava a funcionar, e o objetivo do trabalho era apenas remodelar o visual do *site* conforme o exemplo do *designer*. De início, e por um longo tempo, ainda era muito confuso conseguir achar os elementos necessários, pois o projeto é grande e tem vários componentes. O *Developer* explicou que trabalha fazendo todos os componentes separados, de forma que, se precisa reutilizar algum em outra parte do *site*, é possível.

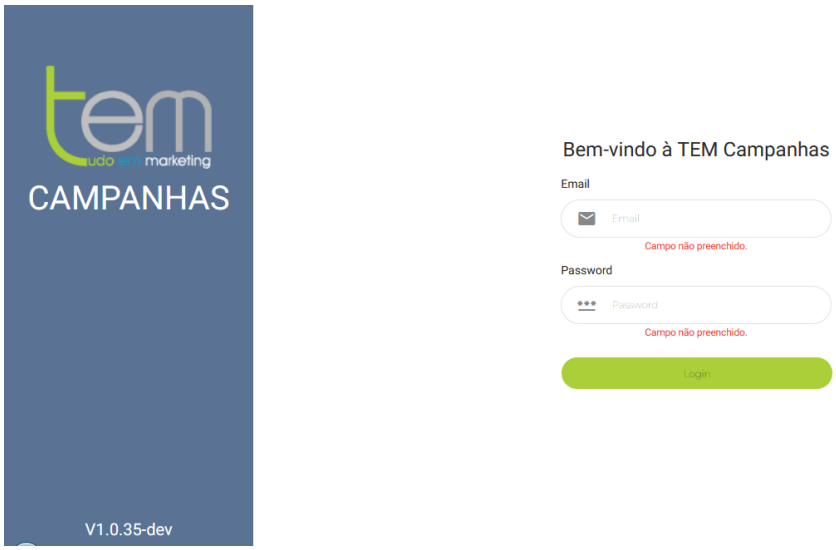


Figura 35: Ecrã de login original

De início, alterei o que já sabia fazer: as cores e locais dos botões. Alterei também o logo. Em certo momento, tive uma experiência mais aprofundada com SVGs: baixei os dois ícones desta página de *login* do ficheiro Adobe XD (o "*user*" e o "*password*"), mas os ícones estavam pretos, e o objetivo era deixá-los cinza. Assim, após alguma pesquisa de como alterar a cor do SVG (testou-se inclusive com o CSS da página, mas sem sucesso), descobriu-se que é necessário alterar o código do SVG dentro do Visual Studio Code mesmo, através do próprio ficheiro. Na imagem abaixo, nota-se que há o "*fill=*" e o código da cor cinza, mas somente no segundo "*path*" dentro do ícone do *user*.

```
<svg id="icon_user_full" xmlns="http://www.w3.org/2000/svg" width="28" height="28" viewBox="0 0 28 28">
  <g id="Group_19" data-name="Group 19" transform="translate(4.656 2.29)">
    <path id="Path_16361" data-name="Path 16361" d="M692.182,71.129c3.177,0,5.55-3.318,5.55-6.285A5.56,5.
    <path fill="#707070" id="Path_16362" data-name="Path 16362" d="M650.973,223.827c3.532,0,6.406,3.143,6
  </g>
  <rect id="Rectangle_13" data-name="Rectangle 13" width="28" height="28" fill="none"/>
</svg>

```

Figura 36: Trocando as cores do SVG

Tal código retorna o ícone da seguinte forma:

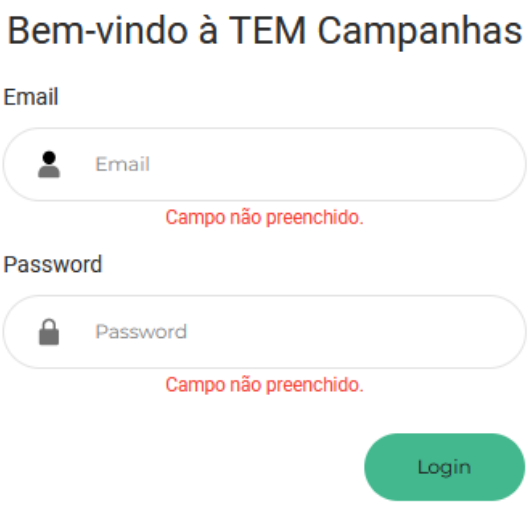


Figura 37: Ícone SVG com cor atualizada

Para deixar o ícone todo cinza, tem de se adicionar um "*fill*" dentro do outro "*path*":

```
<svg id="icon_user_full" xmlns="http://www.w3.org/2000/svg" width="28" height="28" viewBox="0 0 28 28">
  <g id="Group_19" data-name="Group 19" transform="translate(4.656 2.29)">
    <path fill="#707070" id="Path_16361" data-name="Path 16361" d="M692.182,71.129c3.177,0,5.55-3.318,5.55
    <path fill="#707070" id="Path_16362" data-name="Path 16362" d="M650.973,223.827c3.532,0,6.406,3.143,6.
  </g>
  <rect id="Rectangle_13" data-name="Rectangle 13" width="28" height="28" fill="none"/>
</svg>
```

Figura 38: Caminho com todas as cores alteradas

Outro ponto a destacar, que foi uma surpresa, ao trabalhar em um projeto real, todas as cores ganham um nome estático (*primary*, *secondary*), conforme visto na Figura 39:

Figura 39: Cores estáticas

E estes nomes estáticos são depois utilizadas como classes sendo chamadas no HTML (Figura 40), de forma a padronizar o site:

```
<perfect-scrollbar [ngStyle]="{'height': screenHeight + 'px'}">
    <div *transloco="let lang" fxLayout="row" fxLayout.xs="column" fxLayout.lt-md="column" class="body-height no-select">
        <div fxLayout="column" fxFlex="20" fxFlex.xs="100" fxFlex.lt-md="100" fxLayoutAlign="start center" fxLayoutAlign.
        xs="center center" fxLayoutAlign.lt-md="center center" [ngStyle]="{'height':bodyHeight()}"
        class="font-color-white bg-color tertiary-color text-center min-panel-width">
            <object id="logo" data="\assets\img\logo_big.svg"></object>
            <!-- <div fxLayoutAlign="start center" fxLayoutAlign.xs="center start" fxLayoutAlign.lt-md="center start"
            fxLayout.xs="row" fxLayout.lt-md="row">
                <span id="text-logo">{{lang('pages.login.title')}}</span>
            </div> -->
            <div id="version" fxLayoutAlign="end center" fxLayout.xs="row" fxLayout.lt-md="row">V{{version}}</div>
        </div>
        <div fxLayout="column" fxLayout.xs="row" fxLayout.lt-md="row" fxFlex="100">
            <div class="padding-top col-lg-4 col-sm-6 mr-auto ml-auto form-min-width">
                <h1 class="font-color-tem-primary title text-center">{{lang('pages.login.titleWelcome')}}</h1>
                <div class="register-form">
                    <form #form="ngForm" [formGroup]="formLogin" (ngSubmit)="login()">
                        <div fxLayout="column">
                            <label>{{lang('pages.login.login')}}</label>
                            <div class="input-group form-group-no-border" [ngClass]="{'input-group-focus':focus===true}">
                                <div class="input-group-prepend">
                                    <span class="input-group-text">
                                        <span class="material-icons">
                                            <object class="loginicon" data="\assets\img\icons\icon_user_full.svg"></
                                            object>
```

Figura 40: Cores usadas como classe

Desta forma, apesar de ser mais complicado inicialmente para alguém que vê o código de fora, faz mais sentido mais tarde quando há muita repetição. Há de se mencionar, no entanto, que tive um pouco de dificuldades no restante do projeto em achar tais informações, e então criava no CSS de cada componente as cores e alterações que necessitava fazer, pois muitas vezes necessitou-se de modificar um pequeno detalhe de algo, que se mudasse na fonte, modificaria um elemento de outra parte do site que não deveria ser modificado.

A seguir, teve de se modificar o campo *Input*: deixá-lo mais "retangular", ou melhor, diminuir o "*border-radius*". Como no projeto há vários ficheiros SCSS (uma forma aprimorada de CSS), demorou-se um pouco a achar onde estava o código que permitia modificar tal campo:

```
180   }
181   .has-success{
182       .form-control-feedback, .control-label{
183           color: $success-color;
184       }
185   }
186
187   .input-group-append .input-group-text,
188   .input-group-prepend .input-group-text {
189       background-color: transparent;
190       border: 1px solid #5F7391;
191       border-radius: 8px;
192       color: $default-color;
193
194       & i{
195         opacity: .5;
196       }
197
198       @include transition-input-focus-color();
199
200       .has-danger.input-group-focus &{
201           background-color: $white-color;
202       }
203
```

Figura 41: Procura de campos no Visual Studio Code

Para retirar o *placeholder* de dentro dos campos de *input*, foi-se ao código HTML da página de login, e bastou simplesmente remover a menção de mesmo nome, que é uma propriedade pronta do elemento "*input*" do Angular Material (Figura 42):

```
8             <div id="version" fxLayoutAlign="end center" fxLayout.xs="row" fxLayout.lt-md="row">V{{version}}</div>
9         </div>
10        <div fxLayout="column" fxLayout.xs="row" fxLayout.lt-md="row" fxFlex="100">
11            <div class="padding-top col-lg-4 col-sm-6 mr-auto ml-auto form-min-width">
12                <h1 class="font-color-tem-primary title text-center">{{lang('pages.login.titleWelcome')}}</h1>
13                <div class="register-form">
14                    <form #form="ngForm" [formGroup]="formLogin" (ngSubmit)="login()">
15                        <div fxLayout="column">
16                            <label class="font-color-tem-tertiary-extended">{{lang('pages.login.login')}}</label>
17                            <div class="input-group form-group-no-border" [ngClass]="{'input-group-focus':focus===true}">
18                                <div class="input-group-prepend">
19                                    <span class="input-group-text">
20                                        <span class="material-icons">
21                                            <object class="loginicon" data="\assets\img\icons\icon_user_full.svg"></
                                              object>
22                                        </span>
23                                    </span>
24                                </div>
25
26                                <input type="email" required formControlName="user" [disabled]="loading"
                                  class="form-control" placeholder="{{lang('pages.login.login')}}" (focus)="focus=true"
                                  (blur)="focus=false">
27                            </div>
```

Figura 42: *Placeholder*

Com isto, o ecrã de *login* ficou finalizado e praticamente igual ao exemplo feito pelos *designers* (Figura 43):

Figura 43: Ecrã de login atualizado

Após aceder o *site* com Login válido, o mesmo é dividido em 2 outros componentes: "*sidebar*" e "*navbar*". O "*Router-Outlet*" basicamente serve para carregar outros componentes dinamicamente dentro dele, ou seja, irá carregar o componente que for clicado no *sidebar*.

```
1  <div class="wrapper">
2      <div class="sidebar" data-color="tem">
3          <app-sidebar #sideBar></app-sidebar>
4      </div>
5      <div class="main-panel">
6          <app-navbar class="app-navbar-87"
7              (toogleMenuEvent)="sideBar.toogled = $event">
8          </app-navbar>
9          <router-outlet></router-outlet>
10     </div>
11 </div>
```

Figura 44: Divisão do site em *sidebar* e *navbar*

A página de Caixa de Entrada foi escolhida para ser desenvolvida primeiro, já que era a página com mais detalhes feita pelos *designers*. No geral, demorou-se um tempo para achar cada um dos componentes sempre que finalizava algo, mesmo a utilizar a função de Inspecionar do Google Chrome/Firefox.

De início, o plano era modificar os elementos "*Chips*" (Figura 45 - os elementos arredondados perto do topo da página). Para tal, verificou-se o código e tentou-se desvendar onde cada parte se situava nos componentes.

Os *chips* tinham uma imagem em PNG e outras características para cores. Como no ficheiro Adobe XD, cada *chip* tinha uma cor diferente, cada *chip* deveria ter características no código que fossem únicas. No *site* antigo, todos os *chips* se comportavam da mesma forma: quando estavam habilitados, assumiam a cor primária do *site* ao fundo.

Na Figura 45, nota-se também que as cores do menu já foram alteradas da sua versão antiga, e para isto bastou apenas atualizar o código de cores dos elementos estáticos criados para tal (*primary*, *secundary*) do SCSS como dito anteriormente.

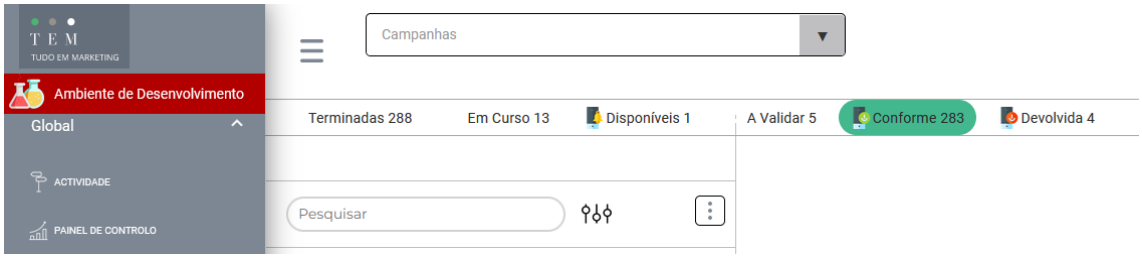
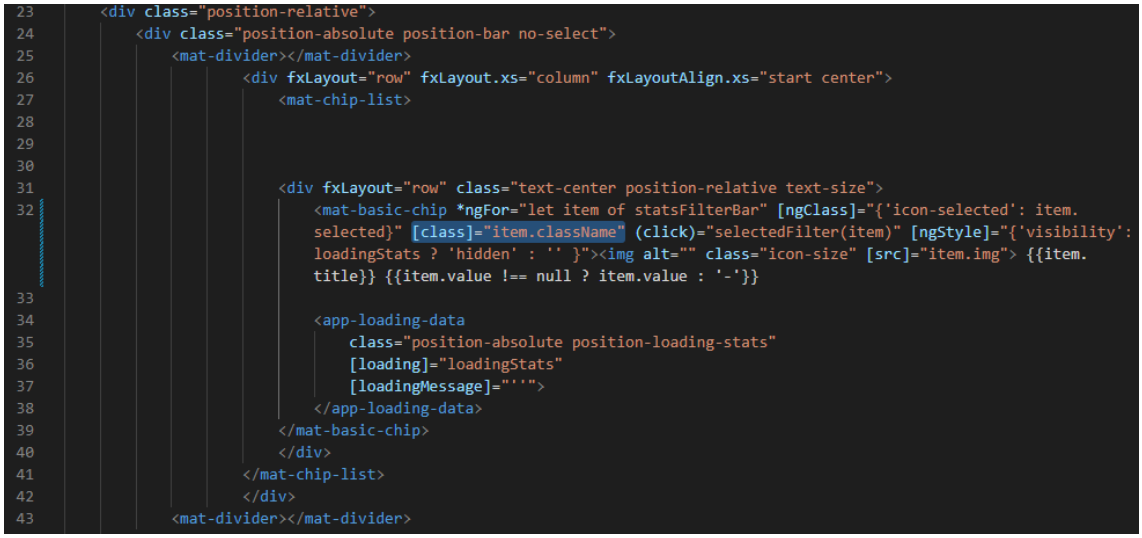


Figura 45: Atualizando os *chips*

Como no Adobe XD, cada *chip* tem uma cor diferente de borda, teve de se pensar em uma forma de fazer cada uma assumir tal característica diferente. Após muito tempo a pesquisar, e insistir que ao usar uma *class* daria para se fazer isso (mas não estava a funcionar com uma classe comum, nem com um "ngIf", nem com um "ngFor"), conseguiu-se chegar na solução de criar uma "[class]", que é um elemento do Angular Material que dá um valor de "*class*" a cada item do "ngFor" (Figura 46):



```
<div class="position-relative">
    <div class="position-absolute position-bar no-select">
        <mat-divider></mat-divider>
            <div fxLayout="row" fxLayout.xs="column" fxLayoutAlign.xs="start center">
                <mat-chip-list>



                <div fxLayout="row" class="text-center position-relative text-size">
                    <mat-basic-chip *ngFor="let item of statsFilterBar" [ngClass]="{'icon-selected': item.
                    selected}" [class]="item.className" (click)="selectedFilter(item)" [ngStyle]="{'visibility':
                    loadingStats ? 'hidden' : '' }"><img alt="" class="icon-size" [src]="item.img"> {{item.
                    title}} {{item.value !== null ? item.value : '-'}}

                    <app-loading-data
                        class="position-absolute position-loading-stats"
                        [loading]="loadingStats"
                        [loadingMessage]="''">
                    </app-loading-data>
                </mat-basic-chip>
                </div>
            </mat-chip-list>
            </div>
        <mat-divider></mat-divider>
```

Figura 46: Elemento [class]

Para que funcionasse, foi necessário criar o elemento "*className*" dentro do "*Array*" de cada *chip* que estava a ser criado com o ciclo "ngFor" (Figura 47):

```
src > app > pages > missions-group-pages > inbox-page > TS inbox-page.component.ts > InboxPageComponent > statsFilterBar > className
    this.mapper.map<Search>('ActionFilter', 'Search', new ActionFilter(null, Tools.getKeyValueOfEnums(ListsTypesEnum.Missi
    this.mapper.map<Search>('ActionFilter', 'Search', new ActionFilter(null, Tools.getKeyValueOfEnums(ListsTypesEnum.Missi
  ];
  columnWidth = '100%';

  statsFilterBar: Array<IconFilter> = [
    {
      img: `${this.pathImage}icon_selected.svg`,
      title: this.translateService.translate('pages.inbox.finished'),
      value: null,
      selected: false,
      type: IconFilterEnum.Finished,
      className: "inboxfinished"
    },
    {
      img: `${this.pathImage}icon_selected.svg`,
      title: this.translateService.translate('pages.inbox.onGoing'),
      value: null,
      selected: false,
      type: IconFilterEnum.OnGoing,
      className: "inboxonGoing"
    },
    {
      img: `${this.pathImage}mobile_bell.png`,
      title: this.translateService.translate('pages.inbox.available'),
```

Figura 47: Elemento *className*

Assim, foi fácil passar cada uma dessas classes para dentro do CSS e alterar, ou seja, cada *chip* pegaria um nome de classe diferente, que estava configurado com cores diferentes no CSS (Figura 48):

```scss
.inboxfinished {
  display: inline-flex;
  height: 35px;
  padding: 7px 12px;
  align-items: center;
  cursor: pointer;
  border-radius: 20px;
  opacity: 1;
  margin-left: 10px;
  background: $theme-tem-secondary-color !important;
}

.inboxonGoing {
  display: inline-flex;
  height: 35px;
  padding: 7px 12px;
  align-items: center;
  cursor: pointer;
  border-radius: 20px;
  opacity: 1;
  margin-left: 10px;
  background: #18B1DE !important;
}
```

Figura 48: Cada chip com um nome de classe diferente

Conseguiu-se, assim, alterar a cor de cada *chip*, e também configurar para terem cor de borda apenas e não mais cor de fundo cinza.

Para os *chips* de missões "Em Curso" e "A Validar", havia o SVG disponibilizado pelo *designer*, mas para os restantes, não havia SVGs para esses *chips*, e houve a necessidade de criá-los alterando um outro SVG disponibilizado em forma de círculo. Para isto simplesmente alterou-se o código até atingir um resultado satisfatório:

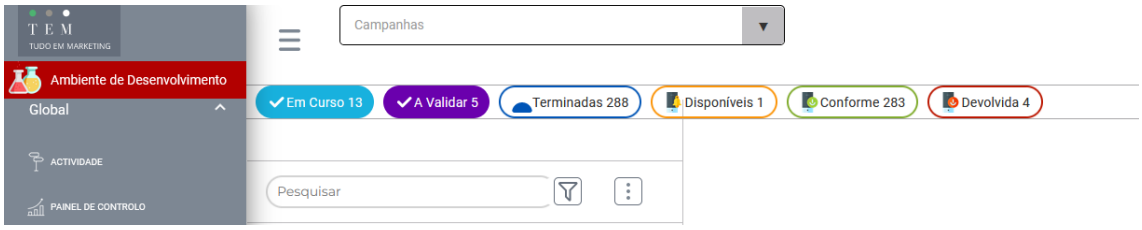


Figura 49: Cores dos chips atualizada

Nota-se que o círculo para o *chip* de "Terminadas" está deslocado, e a característica que define isto é a "*ViewBox*", que altera o local do SVG, e desta forma consegui fazer com que o SVG ficasse centralizado no *chip*.

```
src > assets > img > icons > icon_circle_finished.svg
1  <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="-5 -15 80 80">
2    <g id="icon_circle_finished" transform="translate(-77 -1761)">
3      <g id="Ellipse_28" data-name="Ellipse 28" transform="translate(77 1761)" fill="#fff">
4        <circle cx="24" cy="24" r="24" stroke="none"/>
5        <circle cx="24" cy="24" r="24" fill="#0055B7"/>
6      </g>
7    </g>
8  </svg>
9
```

Figura 50: Elemento *ViewBox*

Após conseguir fazer a bola ficar bem situada, foi uma questão de alterar o CSS dos outros "*className*" que havia criado lá no início para pôr uma cor diferente para cada bola:

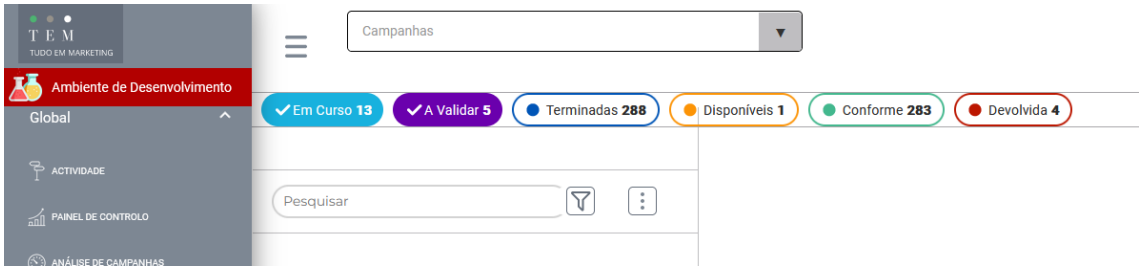


Figura 51: Chips atualizados

Como pode ser verificado na Figura 52 abaixo, no Adobe XD o menu *drop-down* tem uma seta diferente, e tentei modificar ela a procura no código, optando por Inspecionar o elemento pelo Chrome/Firefox – um dos melhores métodos para achar elementos no Angular.

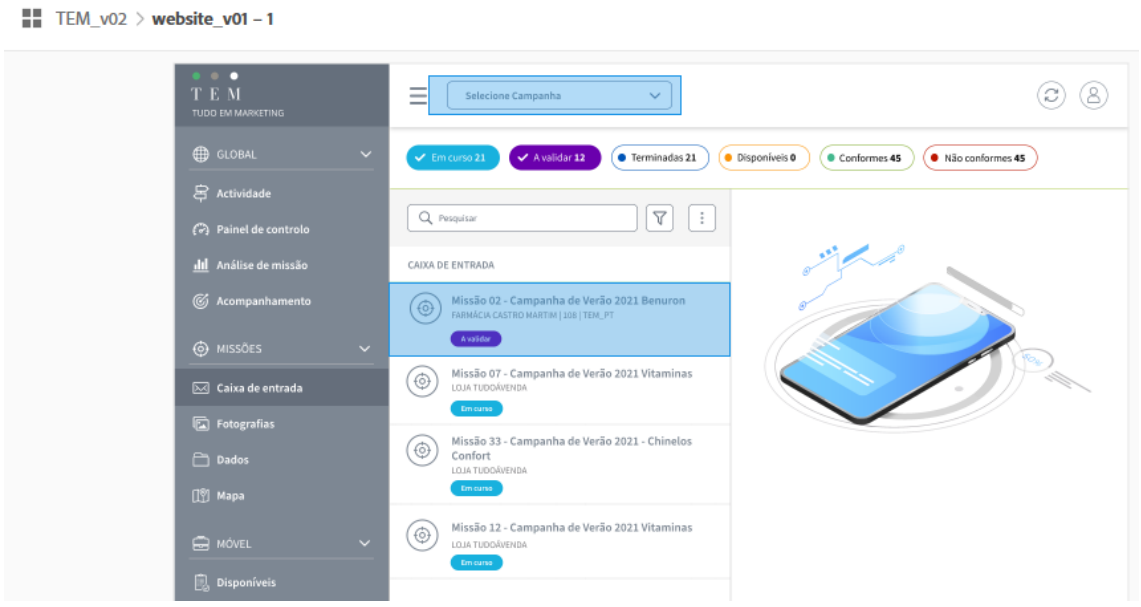


Figura 52: *Drop-down* menu

Conseguiu-se deixar o fundo da seta branco e utilizar a seta SVG proporcionada pelo *designer*, mas sem conseguir retirar a seta antiga que é fixa do elemento Angular Material. Para tanto, trabalhou-se nisto mais tarde.

Para diminuir o espaço entre os *chips* e o campo de pesquisa, o elemento foi Inspecionado pelo *console* do Firefox, e foi achado um *div* "*Header*" (Figura 53). Aprendeu-se mais tarde que, ao utilizar os componentes em mais de uma página diferente, é necessário alterar localmente o tamanho (neste caso a altura) em cada página.

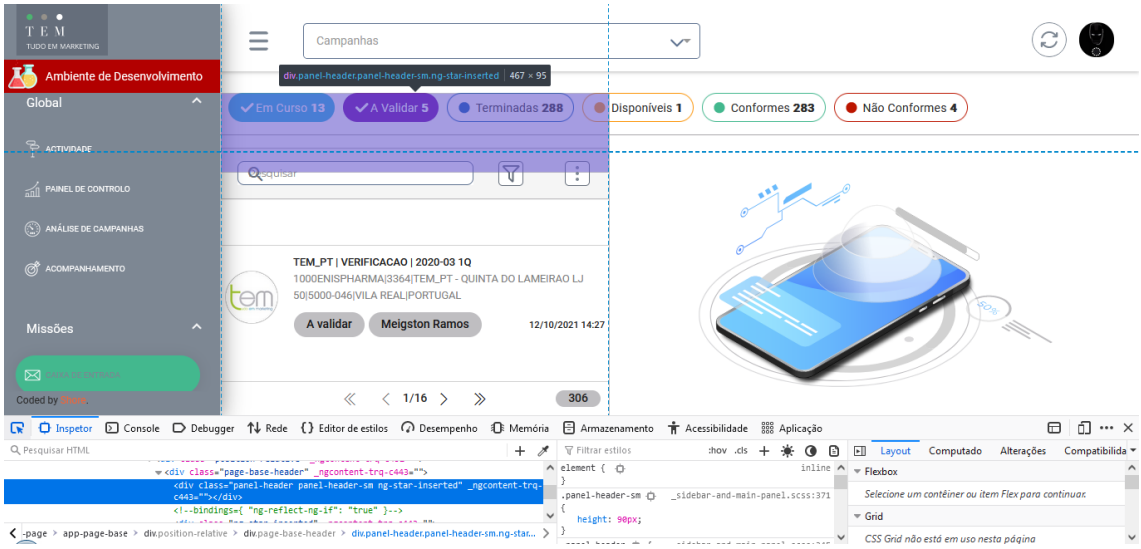


Figura 53: Inspecionando elemento

Ao alterar o CSS da página base (que é reutilizada várias vezes no site), conseguiu-se diminuir o espaçamento daquela *div* "*Header*", acertou-se o posicionamento do ícone de lupa na barra de pesquisa, foi adicionado um fundo mais cinza, e para retirar o traço cinzento que ia além do cabeçalho, simplesmente aumentou-se o tamanho da *div* dos *chips*.

Com esta parte pronta, passou-se o trabalho para o ecrã mais à direita, que aparece quando alguma das campanhas está selecionada.

Foram feitas as alterações dos SVG, posicionamentos, cores e tipos de fonte, como se verifica na Figura 54.

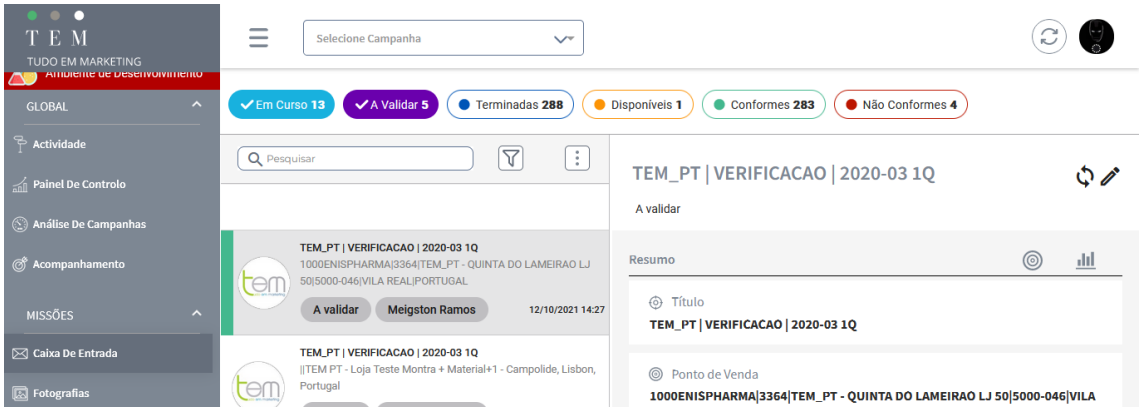


Figura 54: Caixa de Entrada atualizada

Para diminuir o espaço entre as campanhas e a barra de pesquisa, foi alterado o tamanho do elemento "*page-base*" no CSS das missões – visto que a "*page-base*" é utilizada nos outros ecrãs do menu.

O próximo passo, de acordo com o proposto pelos *designers*, seria colocar os ícones ao lado dos menus expansíveis. Para isto, foi procurado no código *TypeScript* pelo elemento *Array* que produzia os títulos, e acabou-se por encontrar o "menu-item.ts", ali, foi criado um novo "*Array*" para as imagens, visto que o código já pegava as informações de um ficheiro JavaScript (Figura 55):

```
src > app > models > TS menu-item-side-bar.ts > MenuItemSideBar > constructor
1   import { RouteInfo } from './core/route-info';
2
3   export class MenuItemSideBar {
4       titleGroup: string;
5       group: Array<RouteInfo>;
6       state: boolean;
7       img: string;
8
9       constructor(title: string = '', items: Array<RouteInfo> = new Array<RouteInfo>(), state: boolean = true, img: string
        = '') {
10          this.titleGroup = title;
11          this.group = items;
12          this.state = state;
13          this.img = img;
14      }
15  }
16
```

Figura 55: *Array* para os itens do menu

O problema que surgiu foi, obviamente, as informações do "*Array*" estão em um ciclo "ngFor", e como foram adicionadas imagens dentro deste ciclo, elas estavam a ser geradas em cada um dos menus expansíveis, e não ficou definido como uma imagem para ser referente a um título.

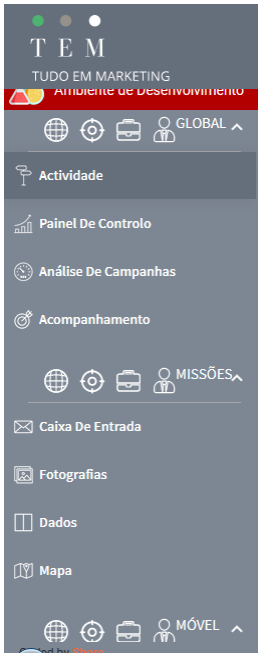


Figura 56: Ciclo de *loop* a criar várias imagens no mesmo título

Para resolver isto, procurou-se uma forma de que cada título fizesse referência a uma única imagem. Foram tentados *loops* dentro de *loops*, e condições "IF" que não deram resultado, mas estava certo de que era uma condição "IF" que faria funcionar. No fim, o problema é que estava a colocar a condição "IF" sem contar os "*indexes*", então ao fazer um "IF" que pegava o número dos elementos do "*Array*" de títulos em um "*index*" para ser igual ao "i*ndex*" dos elementos do "*Array*" de imagens, deu certo (Figura 57).

```
*ngFor="let item of menuItems; index as i"
[expanded]="true"
class="no-bg-color"
(opened)="item.state = true"
(closed)="item.state = false">
<mat-expansion-panel-header class="unique" [matTooltip]="toogled ? item.titleGroup : ''">
    <mat-panel-title class="unique">

        <span *ngFor="let item of menuIcon; index as j"><img *ngIf="i === j" alt="" class="icon-size
        sidebar-icons" [src]="item.img"></span>
        <span class="menu-item-text unique menu-item-texttransf1">{{item.titleGroup}}</span>
    </mat-panel-title>
</mat-expansion-panel-header>
<mat-divider class="divider"></mat-divider>

<span *ngFor="let menuItem of item.group" class="nav-item">
    <li routerLinkActive="active" *ngIf="menuItem.visible" class="{{menuItem.class}} nav-item"
    [matTooltip]="toogled ? menuItem.title : ''">
```

Figura 57: *Index* de elementos

O resultado, no *site*, do ciclo *loop* a funcionar é visto na Figura 58:

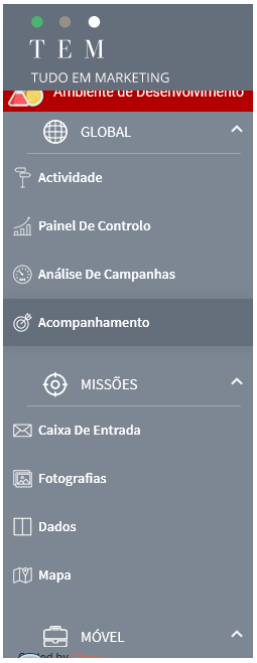


Figura 58: Ciclo ngFor a funcionar

Após isto era hora de atualizar os ícones de cada sessão do menu, e aqui tive ajuda do *Developer* sênior para achar onde estas imagens estavam. Foi necessário modificar cada PNG para SVG, mas não encontrava onde elas estavam no código, e com a ajuda do *developer*, que mostrou que havia feito um componente para que, dependendo de quem tivesse feito *login* (administrador ou utilizador), algumas sessões do menu não apareciam, então tudo estava em um ficheiro de permissões. Ao localizá-lo, bastou fazer referência aos novos SVG e posicionar e alterar o tamanho conforme apropriado (Figura 59).

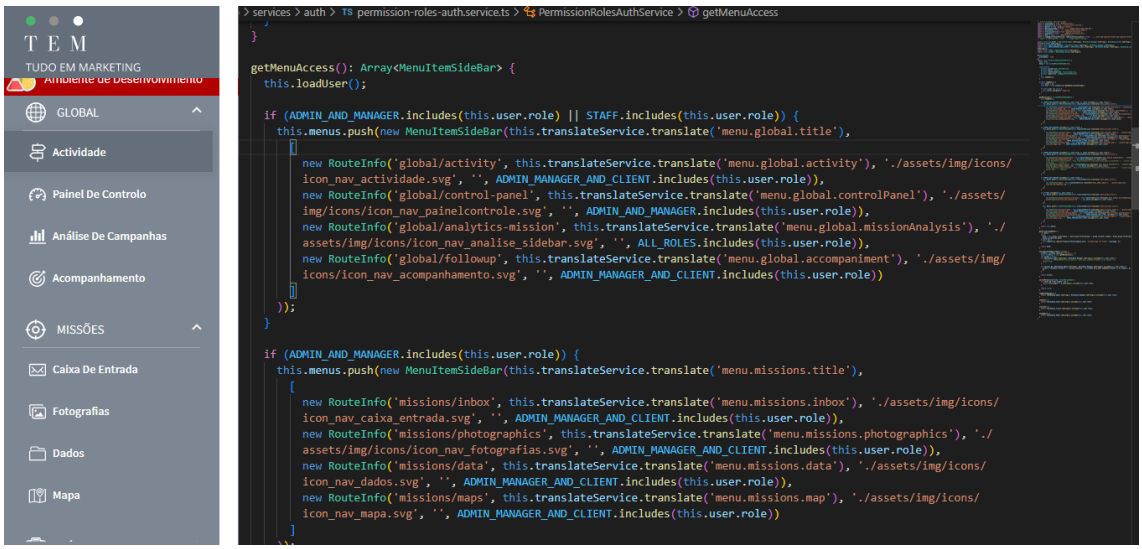


Figura 59: O menu atualizado e o local dos ícones no código

Com isto feito, no último dia do estágio foram realizadas outras pequenas modificações no *layout* conforme indicado pelo *designer*, inclusive ao adicionar um título a caixa de entrada.

Figura 60: Resultado no fim do estágio

5. Conclusão

Com este relatório, pôde-se fazer um apurado de tudo que foi realizado nos meses de estágio realizado na Shore Spun Lda. Houveram muitas outras situações menores que impactaram positivamente o aprendizado, e muito o que aprendeu-se deve-se à paciência e prestatividade em ensinar do desenvolvedor sênior que supervisionou o estágio mais de perto, de forma mais ativa.

Um dos pontos de maior importância foi que no decorrer do estágio foi necessário aprender a utilizar ferramentas, *Frameworks*, bibliotecas e conceitos totalmente novos que não foram vistos durante a graduação, o que é extremamente positivo, no sentido de que o estágio foi um ótimo complemento para o que já havia sido visto, além de dar uma experiência de trabalho em uma empresa real, com necessidades reais e problemas reais, além do óbvio trabalho em equipa e necessária comunicação para fazer tudo dar certo.

É certo que houveram várias dificuldades no caminho. No começo, o estágio começou um tanto lentamente, então houveram períodos em que parecia que não havia um plano para atividades que eu pudesse fazer. Segundo, muitas vezes tive de pesquisar e procurar por soluções por conta própria, onde gostaria de ter tido uma mentoria mais próxima. Em relação a isso, no entanto, não há de se reclamar muito, porque é compreensível que os outros colaboradores da empresa tivessem suas próprias preocupações e não pudesse auxiliar o estagiário.

Somado à essa necessidade de busca por respostas sozinho, o fato de ter de aprender Angular no início do estágio sem ter tido qualquer menção deste *framework* durante a graduação, foi um desafio imenso. Por mais que os conceitos de programação sejam os mesmos, a forma de trabalhar e a própria natureza do código produzido através do Angular é diferente e único, portanto muitos conceitos demoraram a ser assimilados, principalmente por ter de depender de tutoriais de *youtube* para aprendê-los.

Neste mesmo sentido, durante o projeto principal que foi atribuído, houve muita dificuldade às vezes em relação a simplesmente achar os elementos dentro do código. Ao meu ver, tal dificuldade se dá pelo fato de ter sido dado um projeto de um *site* inteiro que já estava funcional, com todos os seus elementos, e a missão de modificar os elementos para ficarem como a equipa de *design* definiu.

O sentimento que tive durante a maior parte do estágio foi de que os conteúdos eram extremamente complexos (aqui, possivelmente também por ser o primeiro contato com tais conteúdos, como é normal em qualquer tópico), e que teria de revisitar os vídeos de *youtube* e tutoriais dos *sites* oficiais, bem como a documentação, para efetivamente aprender. No fim, penso que a máxima de que a prática traz a perfeição está correta, pois somente com a prática se adquire a velocidade e o entendimento necessário para codificar efetivamente.

Muito tempo também foi disposto para resolver problemas de código, mas como dito acima, foi problema de não ter o conhecimento e a experiência de alguém que conhece e trabalha com Angular há anos. Às vezes, fui pego por problemas de iniciante, como o código não funcionar simplesmente por ter faltado importar algum módulo do Angular Material, por exemplo.

De uma forma geral, o objetivo de um estágio foi satisfatoriamente concluído. Aprenderam-se habilidades e ferramentas que antes não se dispunham, conheceu-se novos *frameworks*, linguagens e formas de trabalhar, além da inestimável experiência de se trabalhar dentro de uma empresa consolidada com clientes e projetos reais. Ao fim do estágio, pode-se dizer com tranquilidade que consigo criar um *site* com a utilização de Angular e suas bibliotecas, deixando-o funcional e moderno, porque a experiência percorrida durante o estágio, em suas diversas fases e projetos atribuídos, deram-me o conhecimento necessário para atingir tal fim.

6.Referências Bibliográficas

(1) **Shore | LinkedIn.** [Consult. 12 Setembro 2022]. Disponível na WWW: URL: https://pt.linkedin.com/company/shore-spun.

(2) **Shore | Consulting.** [Consult. 12 Setembro 2022]. Disponível na WWW: URL: https://www.shore.pt/.

(3) **loiane.training** [Consult. 02 Setembro 2022]. Disponível na WWW: URL: https://loiane.training/curso/angular.

(4) **Angular Material: como instalar e usar?** [Consult. 07 Setembro 2022]. Disponível na WWW: URL: https://blog.betrybe.com/framework-de-programacao/angular-material/.

(5) **Angular Material UI componente library.** [Consult. 07 Setembro 2022]. Disponível na WWW: URL: https://material.angular.io/.

(6) **O que é API? Canaltech.** [Consult. 09 Setembro 2022]. Disponível na WWW: URL: https://canaltech.com.br/software/o-que-e-api/.

(7) **Serviços de computação em nuvem.** [Consult. 09 Setembro 2022]. Disponível na WWW: URL: https://cloud.google.com/.