Relatório Final de PDS

8 de junho

BetterCode

Da autoria de: André Freitas 21112
André Freitas 21126
Iuri Rodrigues 21159
Luís Silva 21113
Nárcio Silvestre 21723

IPCA

Sobre o Projeto

Quem somos?

A nossa empresa passa pelo nome de BetterCode

Ela é composta por cinco elementos com cargos diferentes

- André Freitas – Productowner
- André Teixeira – Scrum Master
- Iuri Rodrigues – Development Team Member
- Luís Silva - Development Team Member
- Nárcio Silvestre - Development Team Member

Problema a abordar?

O nosso projeto “BarberShop Management System”, irá consistir na gestão de uma franchise de barbearias onde será possível ter acesso a todos os serviços de barbearia bem como serviços de tatuagem.

Regras de negócio

Foram definidas uma série de regras de negócio para ajudar no planeamento do projeto:

- O sistema deve guardar a informação dos clientes: nome, NIF, morada, código postal, email, número de telemóvel.
- Este deve guardar a informação dos barbeiros: nome, NIF, número de telefone, email.
- A barbearia oferece vários tipos de serviço (nome, custo) com custos variados.
- Entre estes serviços, existem dois tipos de serviços primários: barbearia e tatuagem.
- O sistema após o pagamento emite um recibo. O cliente pode pedir ou não fatura com NIF.
- É possível fazer uma subscrição a um plano VIP mensal.
- Esse plano terá uma série de vantagens e regalias a definir.
- Cada barbeiro tem uma agenda de trabalho. Cada cliente pode agendar várias sessões com o mesmo barbeiro ou com um barbeiro diferente.
- Quando ocorre um imprevisto, o barbeiro cancela a marcação e o sistema envia um email para o cliente a informar do cancelamento.
- O mesmo se aplica para o cliente, aí o barbeiro receberá uma notificação do sistema a informar do cancelamento.
- No site, o cliente pode ver as barbearias que estão perto do local onde moram.
- O cliente pode mudar o seu perfil no site.
- Cada barbearia tem o seu horário

Diagramas UML

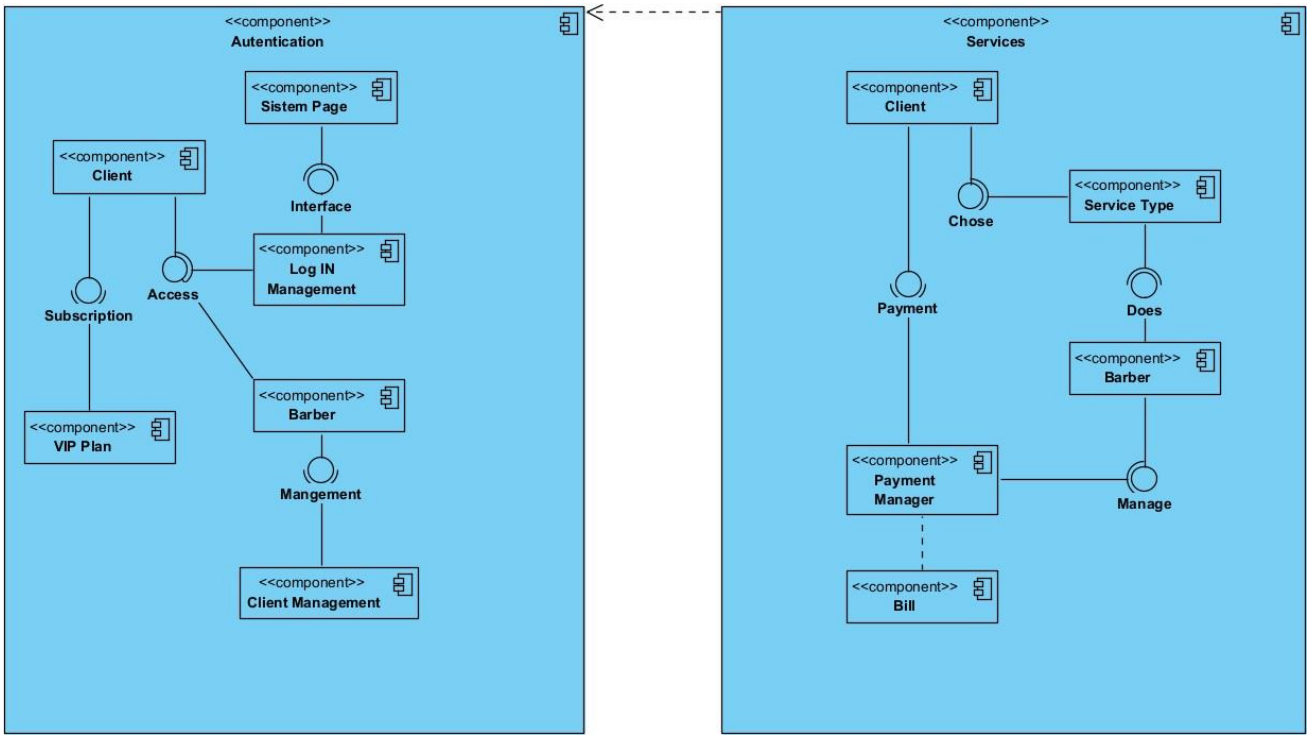


Diagrama de componentes

Nesta figura encontra-se um diagrama de componentes que é composto por dois componentes principais, a autenticação e os serviços. Ao mesmo tempo, estes dois componentes são ambos compostos por subcomponentes.

O primeiro componente “Autentication” representa o funcionamento do Log IN do sistema. Começamos por entrar no site (“Sistem Page”), de seguida irá aparecer uma tela a pedir o email e a password (“Log IN Mangement”), a partir dessa tela será possível escolher o cargo do utilizador, se ele é um cliente ou um barbeiro. Caso o utilizador aceda a área de barbeiro este poderá ter acesso à gestão de cliente nomeadamente as suas marcações. Caso ele aceda a área de cliente irá ter acesso à subscrição de um plano VIP.

O segundo componente “Services” retrata o cumprimento de um serviço. Este componente também envolve o cliente e o barbeiro simultaneamente. O cliente começa por escolher o tipo de serviço (“Service Type”), ao mesmo tempo o babeiro irá interagir com esse componente dado que é ele que faz o serviço, depois disto o cliente procederá ao pagamento (“Payment Manager”) enquanto isso o barbeiro irá receber o pagamento. Para acabar depois do pagamento já feito o cliente decidirá se irá querer fatura ou não.

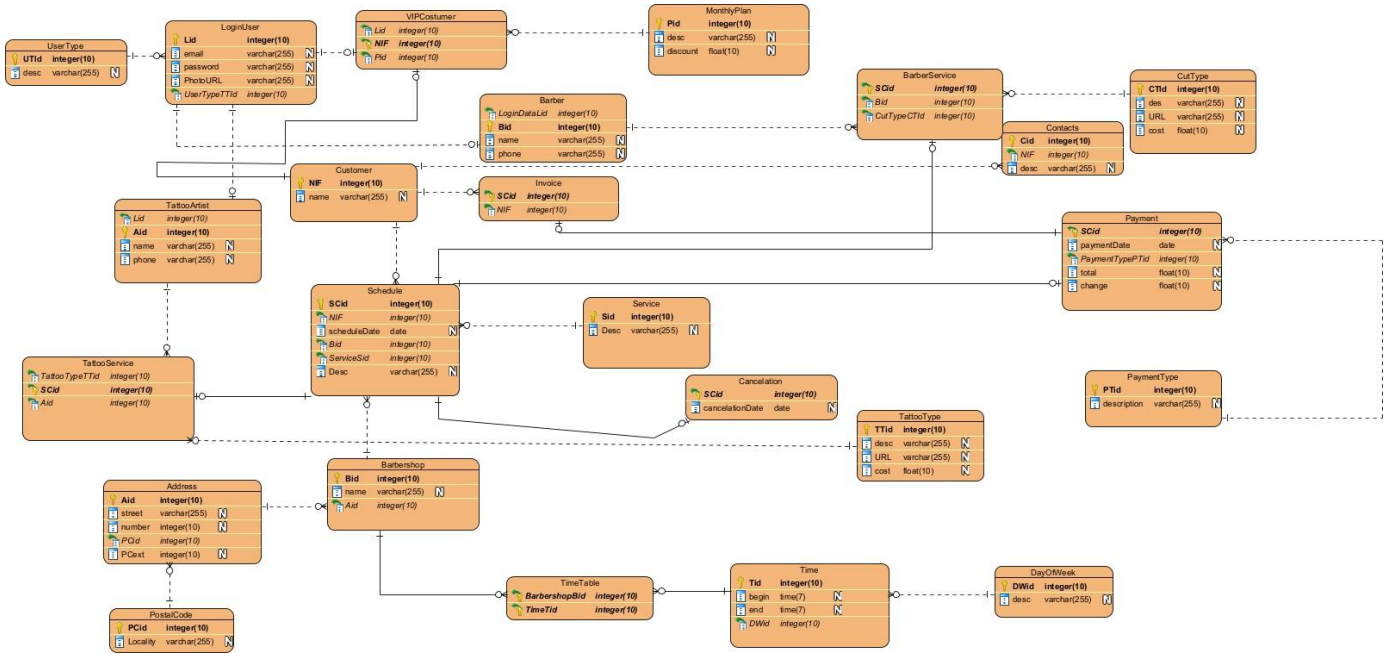


Diagrama de Base de Dados

Em cima encontra-se representado o diagrama ER do projeto. Este diagrama é composto por 27 tabelas diferentes todas ligadas entre si por relações de: um para muitos, muitos para 1, 0 ou mais para muitos e por fim relações de um para um.

Começamos com a tabela de Login na qual será possível armazenar o email, a password e o tipo de utilizador, algo que faz com que exista uma ligação a outra tabela responsável pela definição do tipo de utilizador. Além disso é possível aceder a tabela de "VIP Costumer", esta é conectada à tabela de planos mensais.

A tabela do barbeiro além de ligada à tabela de "Log IN", permite aceder a tabela de serviço de barbearia e consequentemente a tabela de tipo de cortes de cabelo e barba. Exatamente o mesmo se aplica à tabela do tatuador, onde o processo é o mesmo embora em vez de cortes de cabelo sejam tatuagens. Por fim depois do log in também é possível chegar à tabela do cliente.

Estas três tabelas vão ligar a uma tabela de agenda ("Schedule"). Posteriormente esta permite aceder a uma tabela de serviço generalizada, bem como uma de cancelamento dos próprios serviços.

Em conjunto com a tabela "Costumer" a tabela de agenda irá conectar-se a uma tabela de pagamento, que permite ainda o tipo de pagamento.

Finalizando, a tabela "Barbershop" contém o nome e a morada de uma barbearia, morada esta que é contida numa tabela à parte, mas que está na mesma ligada à tabela da barbearia. Além disto, a barbearia irá suportar uma tabela de horário, composta pela tabela de tempo e a tabela do dia da semana.

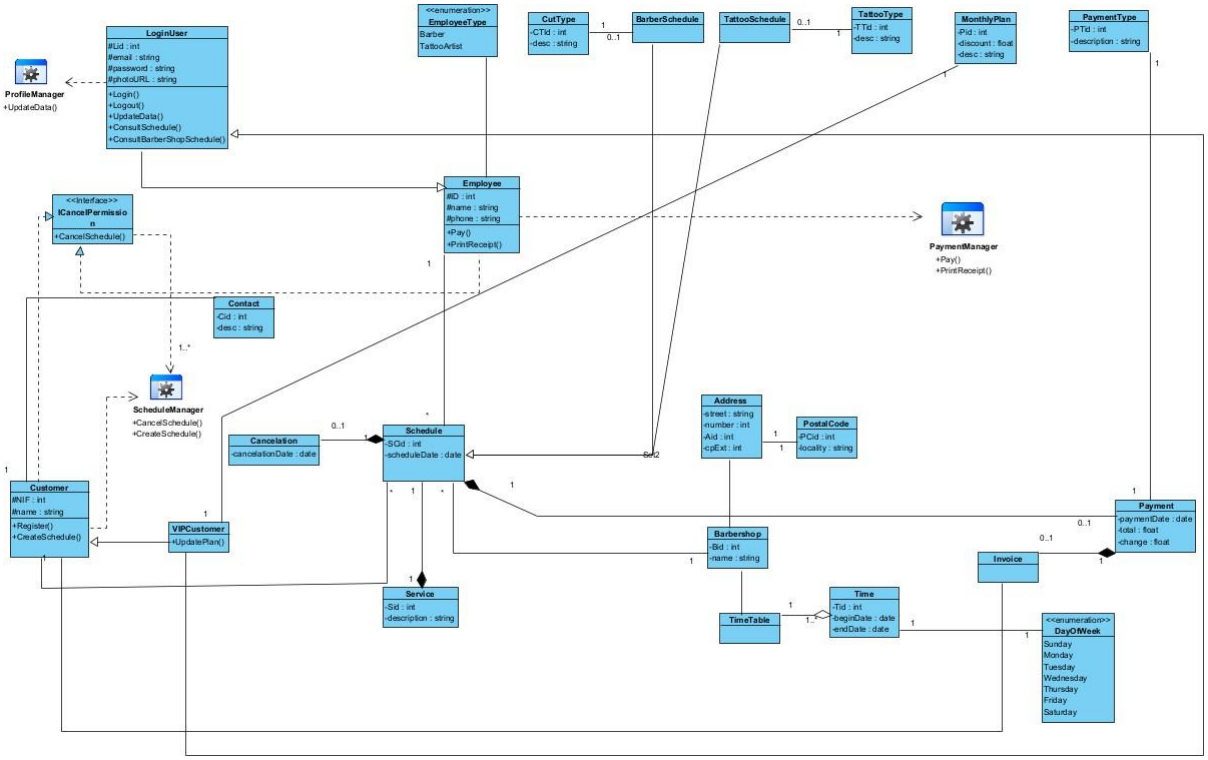


Diagrama de classes

Semelhante ao falado em cima sobre o diagrama de tabelas de dados, este diagrama contém 27 classes, sendo de cada classe atua de forma muito semelhante à sua contraparte no diagrama ER.

Sendo assim seria mais acertado focar nas diferenças entre os dois.

Em primeiro lugar, ao contrário de existir duas tabelas diferentes para os funcionários, sendo estas o barbeiro e o tatuador, aqui existe apenas uma classe generalizada de funcionários. Esta classe de funcionários está diretamente ligada à classe de cancelamento de um serviço, algo diferente daquilo que se verifica no diagrama de tabela de dados.

Em segundo lugar existe duas classes para a agenda, uma para cada tipo de serviço primário (barbearia e tatuagem), em contraste da única tabela criada para a agenda no diagrama ER.

Por fim a classe da morada é estendida a uma classe de código postal pois este não poderia estar dentro de uma classe dado que além de variáveis inteiras, este contém um hífen na qual não pode ser incluído.

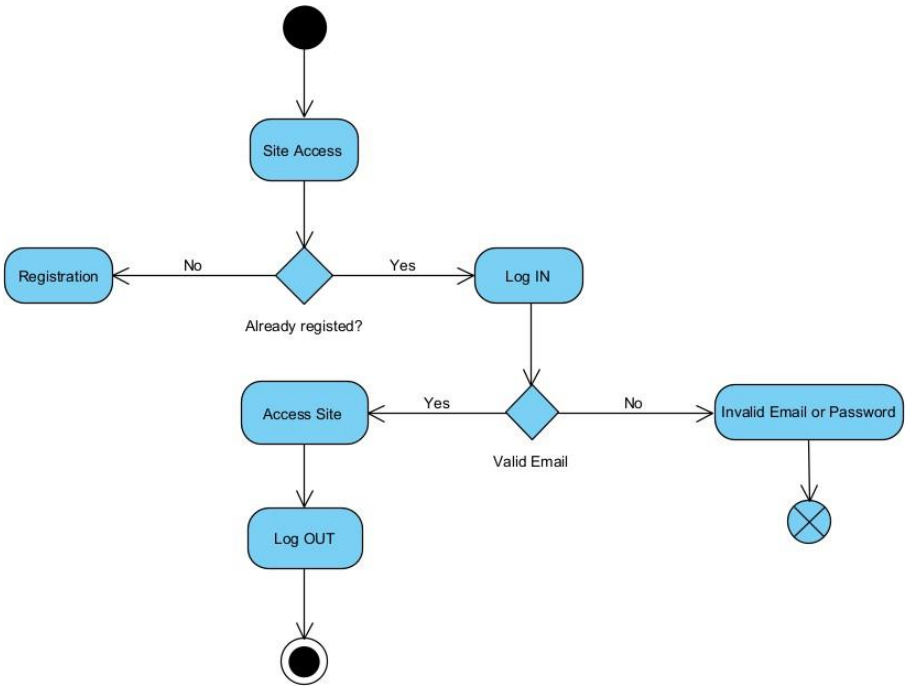


Diagrama de atividades sobre a Autenticação

O diagrama de atividades relativo ao sistema de Log IN do site é bastante simples. Ao aceder o site existe uma verificação perante a existência de conta ou não. Caso não exista o utilizador regista-se, caso exista este faz log in com a conta registada. Após isto é verificada a validade do email e da password, caso alguma delas não corresponda com a certa, um erro irá aparecer que fará o utilizador tentar outra vez.

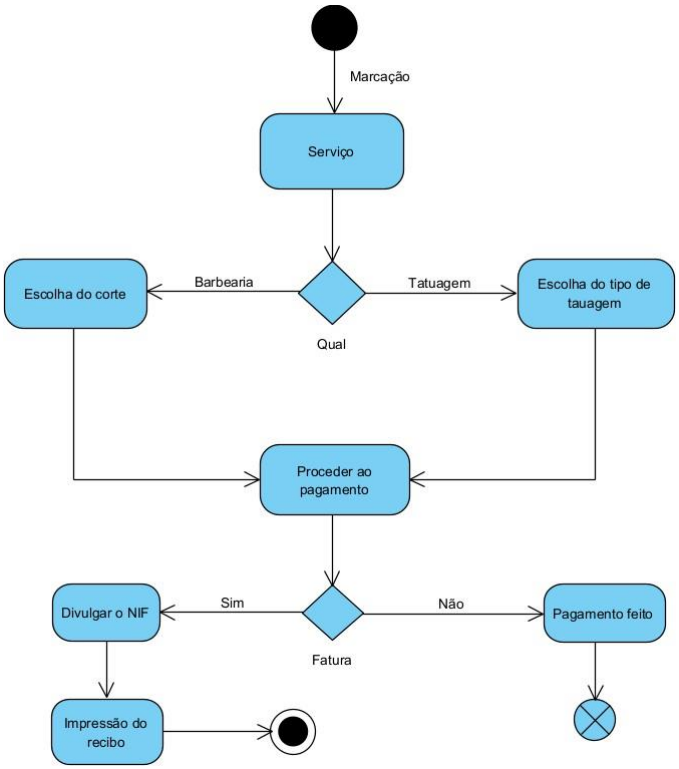


Diagrama de atividades representativo dos serviços

Aqui encontra-se apresentado o diagrama de atividades que representa o funcionamento de uma marcação.

O cliente começa por escolher qual dos dois tipos de serviços primários ele quer, sendo eles a barbearia e tatuagem. Após isto o cliente terá de escolher qual o corte ou qual a tatuagem dependendo de escolha anterior.

Dado como acabado o serviço, o cliente tem a opção de pedir fatura com contribuinte, ou não. Se não pedir o pagamento fica concluído, se quiser irá receber um talão impresso pelo barbeiro.

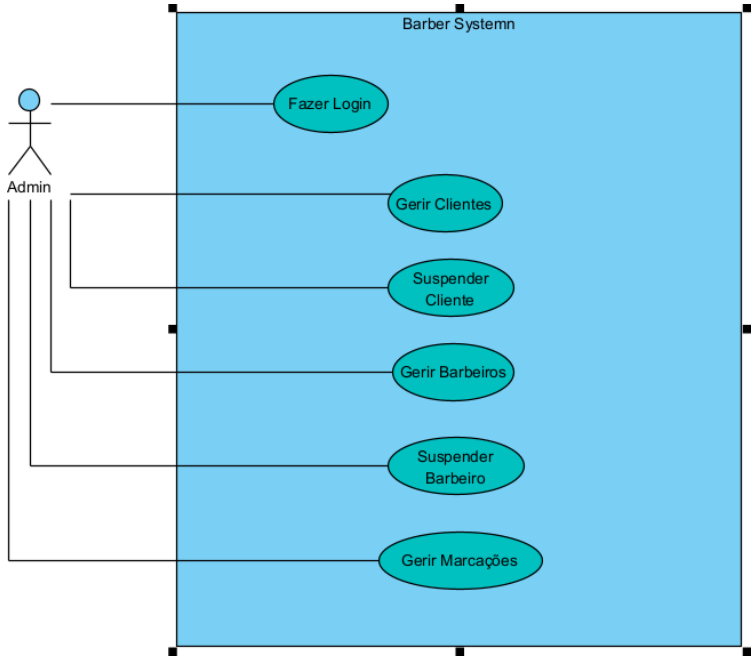


Diagrama de Caso de Uso do Administrador

Este diagrama de caso de uso retrata tudo aquilo que o administrador será capaz de fazer.

Para começar ele poderá fazer login no site e a partir daí ter um ramo de opções na qual este poderá escolher. Além de gerir as marcações, isto é, atualizá-las ou cancelá-las, o administrador poderá também gerir os clientes e os barbeiros.

Com isto ele poderá suspender subscrições VIP, bem como demitir barbeiros e dar folgas.

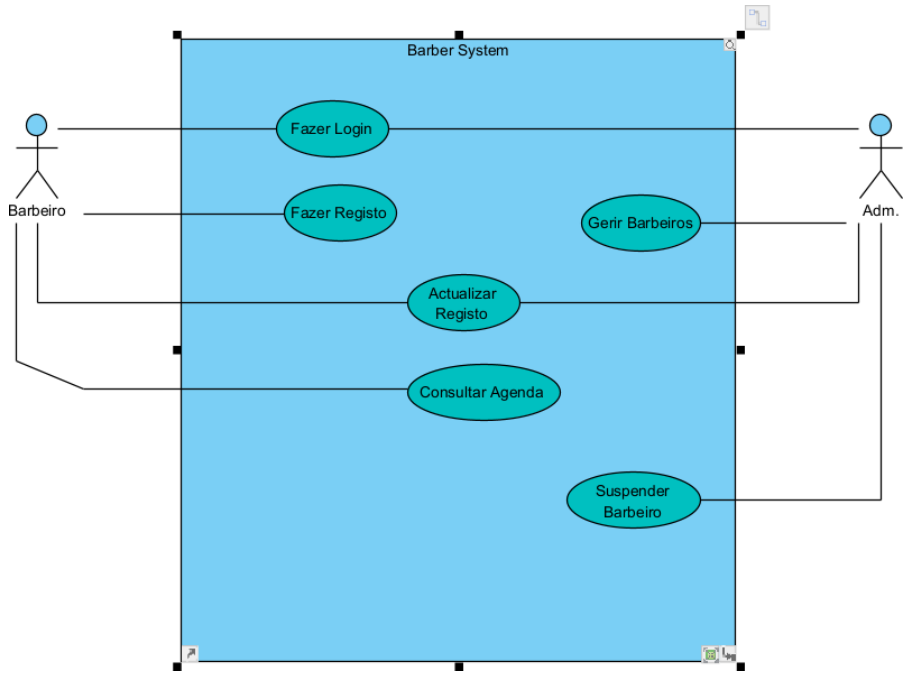


Diagrama de Caso de Uso do Barbeiro

A figura acima retrata aquilo que um barbeiro tem acesso no site, além de incorporar as coisas que o administrador pode fazer, que afetam diretamente o barbeiro.

O barbeiro terá acesso à agenda de marcações e ele também pode influenciar e até mesmo alterar estas marcações caso acha que não poderá corresponder a marcação definida.

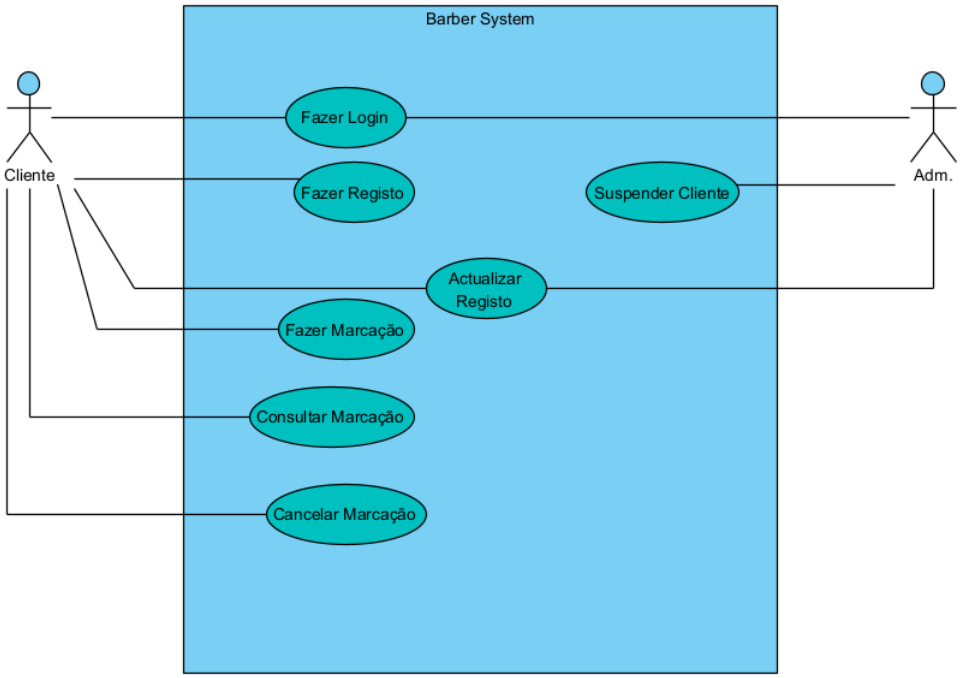
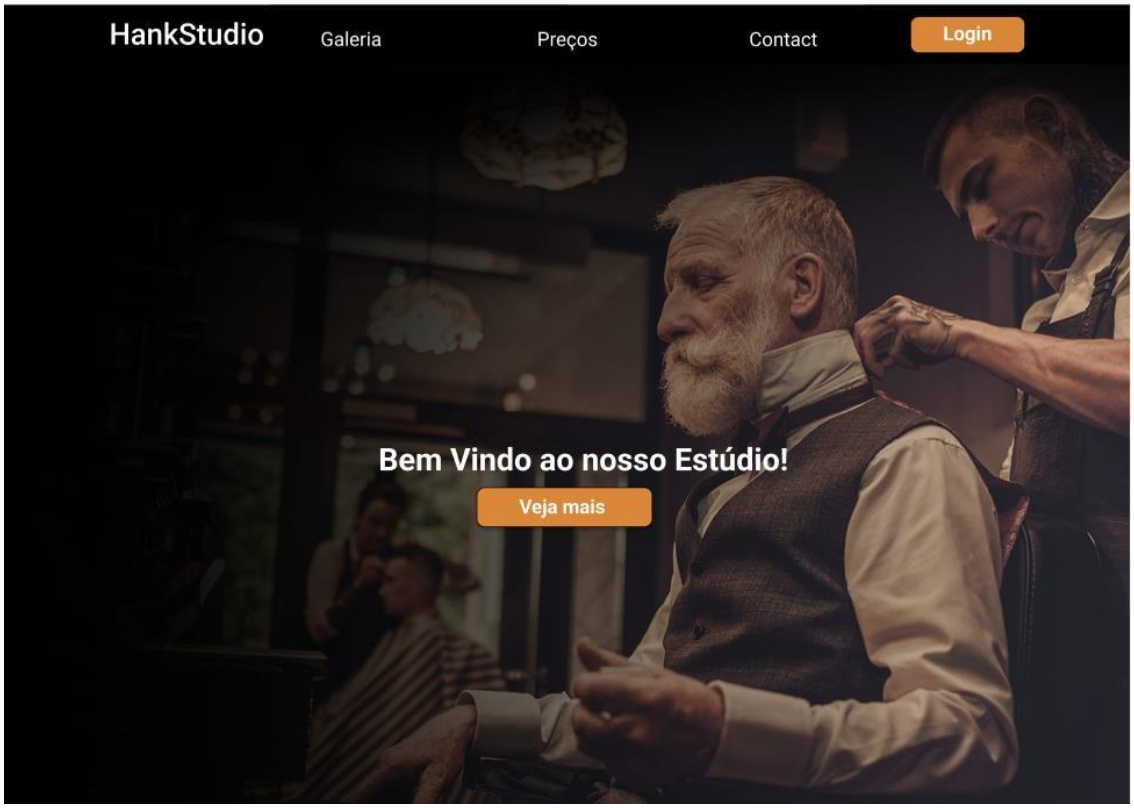


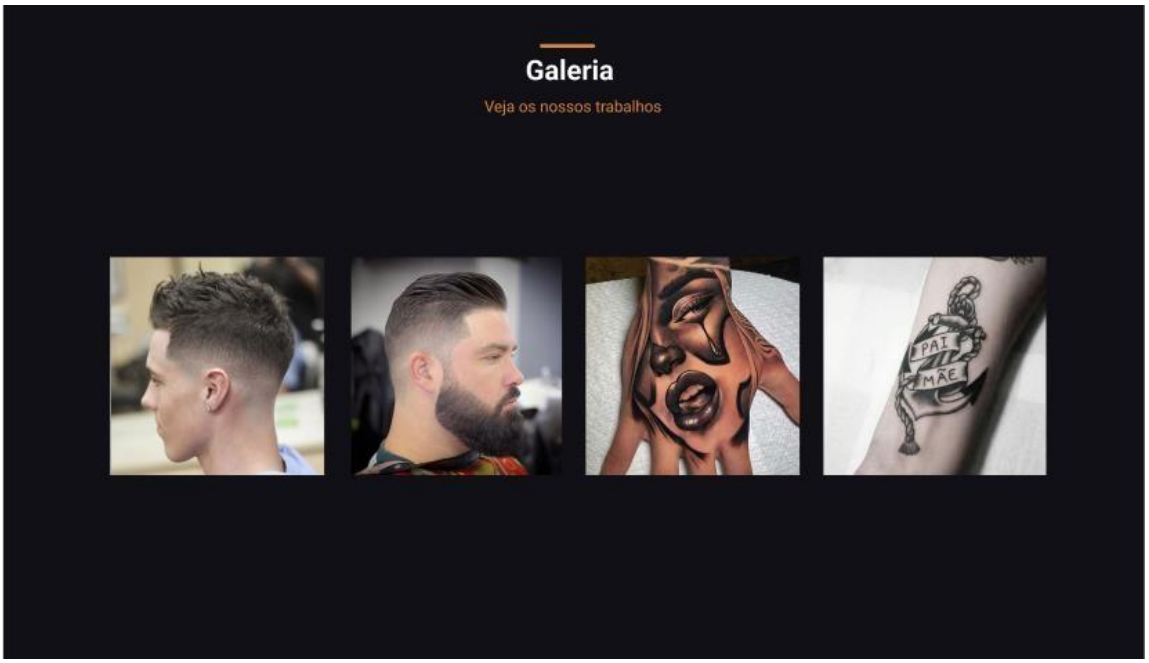
Diagrama de Caso de Uso do Cliente

Por último temos um diagrama de caso de uso representa tudo aquilo que o cliente poderá fazer no site. Começando pelo básico este poderá registar-se e fazer log in, como qualquer um dos outros utilizadores. Dito isto é claro que este também tenha domínio sobre as marcações que efetua seja só para consultá-las ou então cancelá-las.

Mockups dos sites

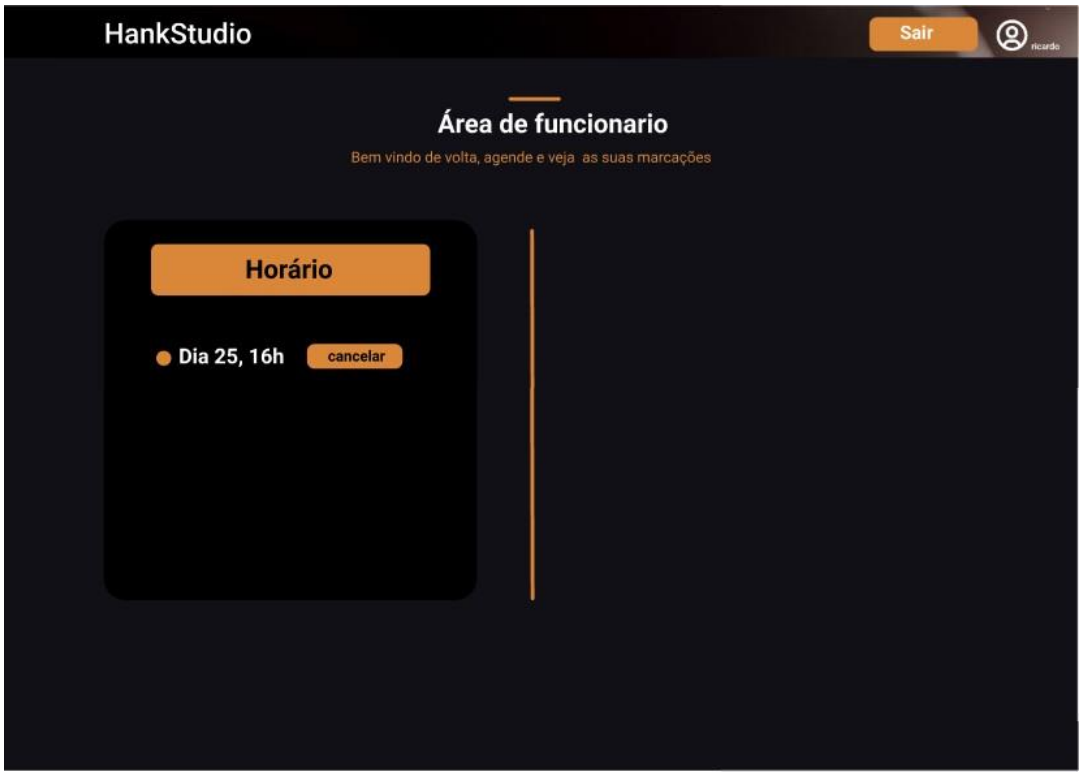


Mockup da página inicial

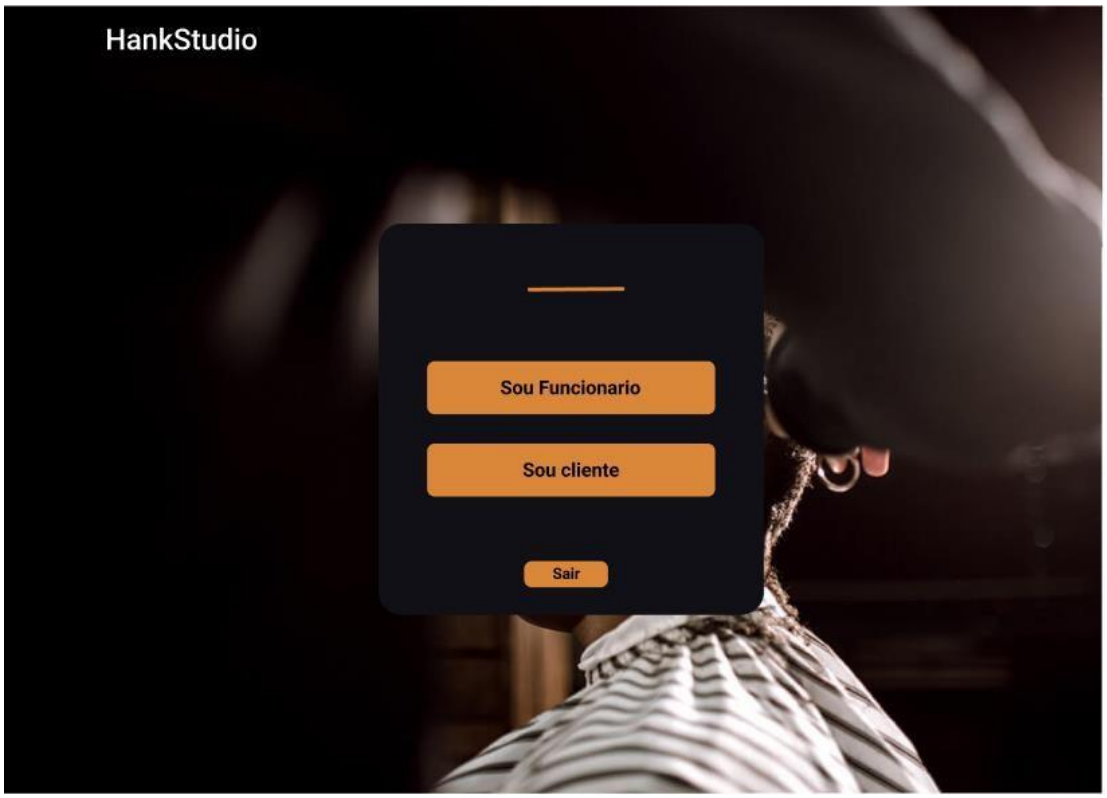


Mockup da Galeria

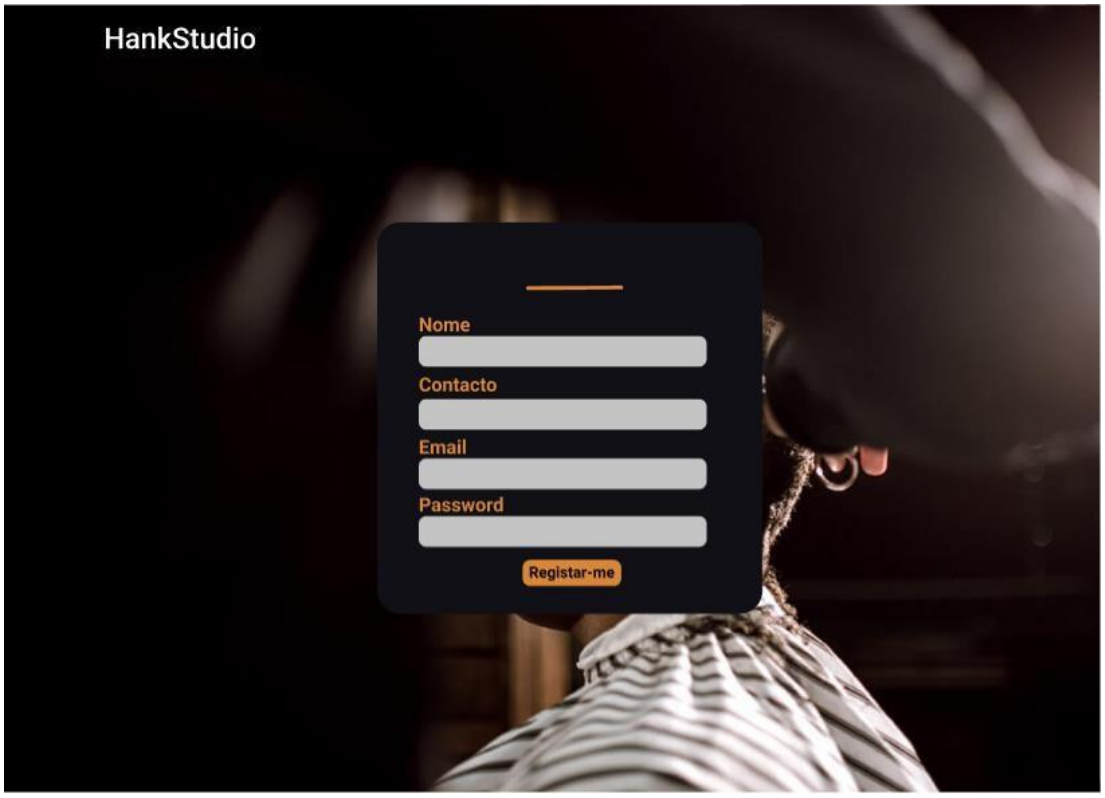
Mockup da Tabela de Preços



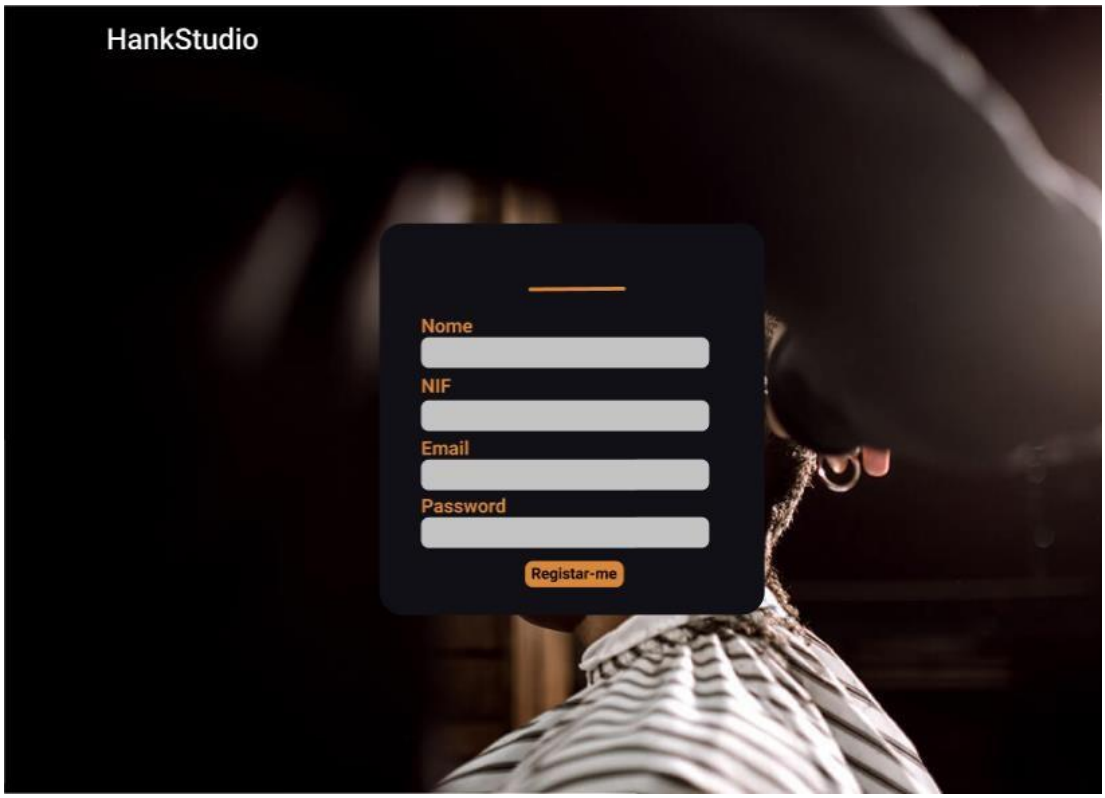
Mockup da Área do Funcionário



Mockup do Login

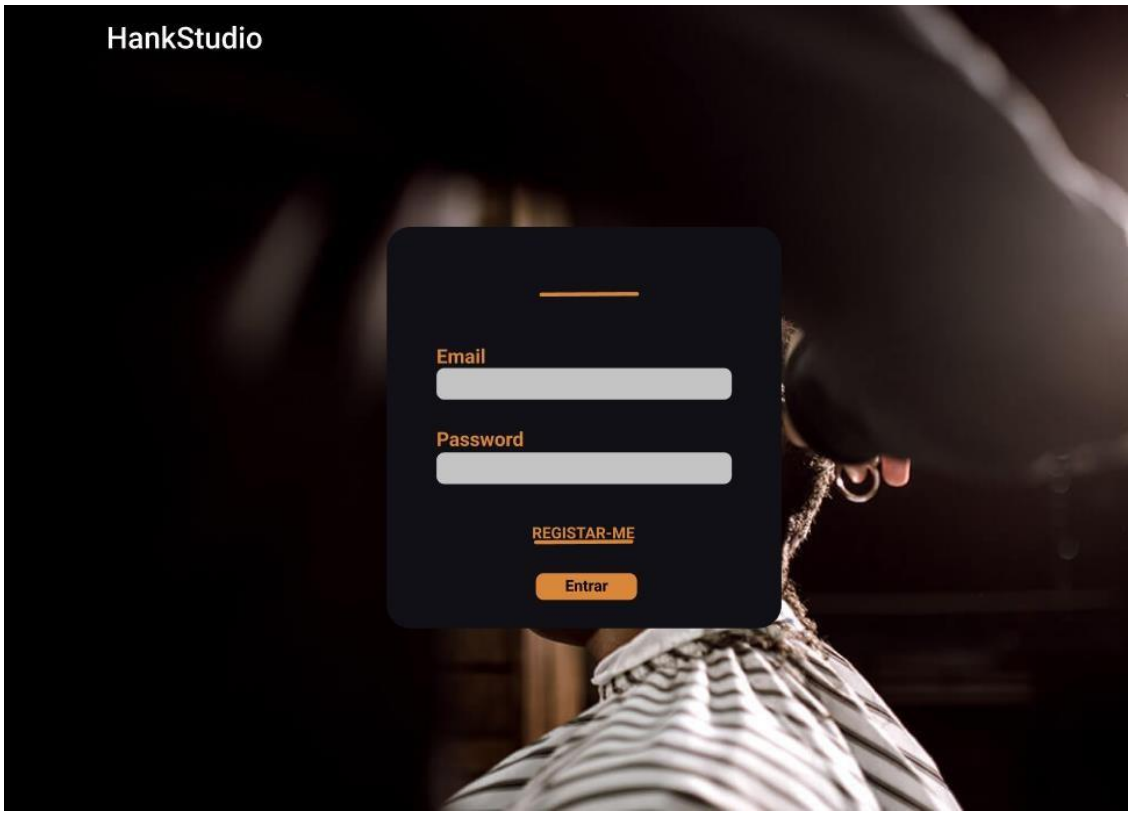


Mockup de Registo de Funcionário



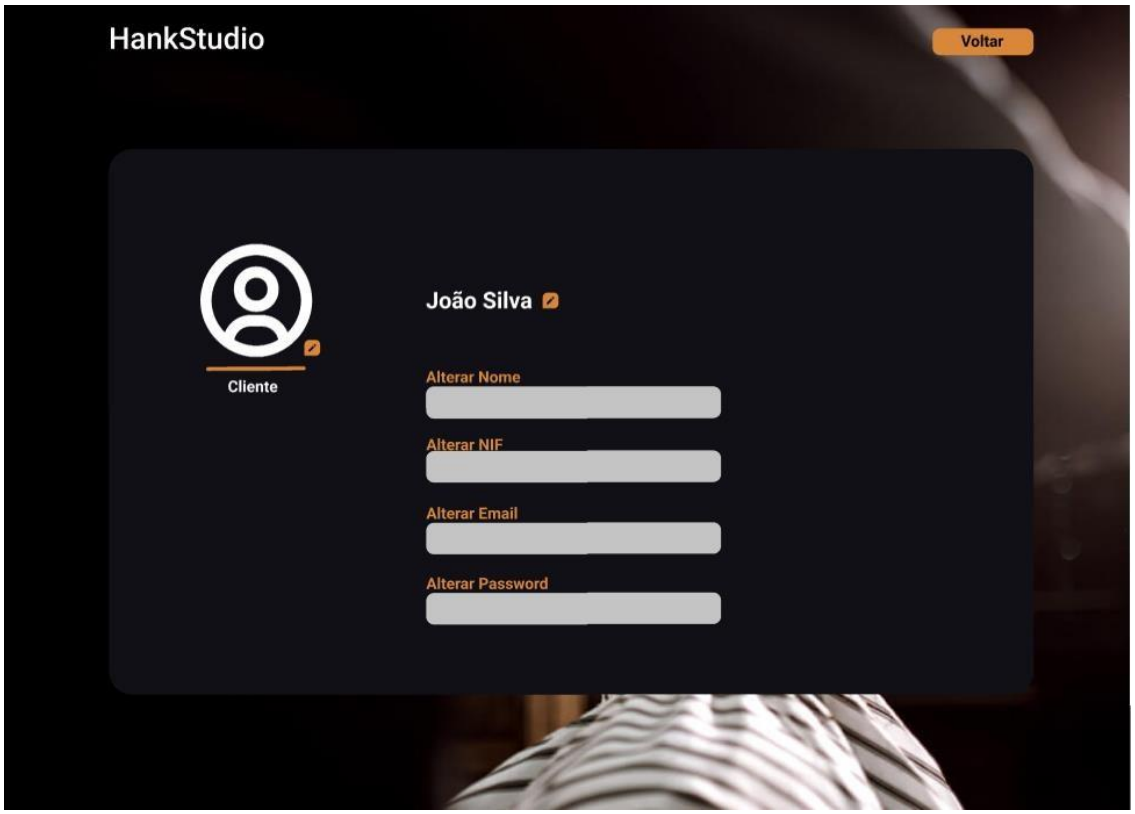
Mockup de Registo de Cliente

Mockup da Área de Cliente

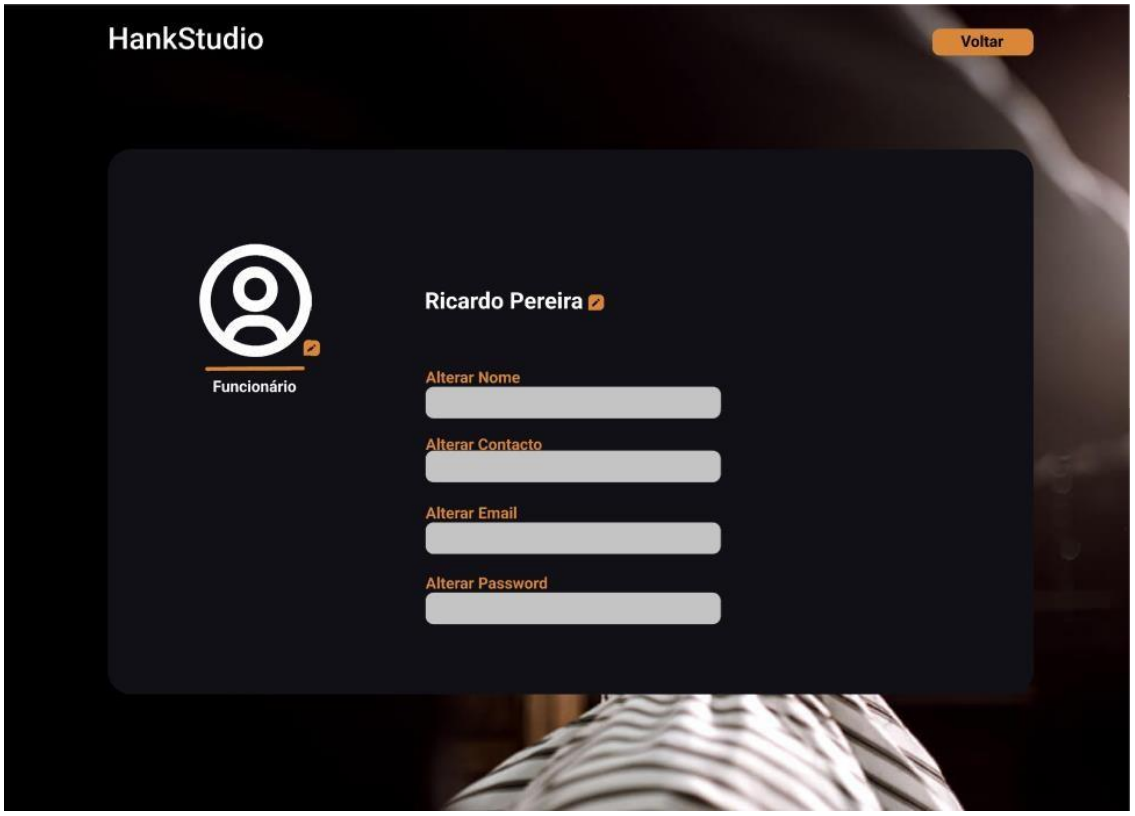


Mockup de Login de cliente/ funcionário

Mockup da Aba de Contactos



Mockup de Edição de Perfil do Cliente



Mockup de Edição de Perfil do Funcionário

Base de Dados

Para podermos armazenar todas as informações e todos os dados necessários para o projeto, usufruímos de uma base de dados MySQL.
Para poder aceder e trabalhar na mesma, foi necessário usar o WorkBench disposto pelo MySQL.

De modo a podermos trabalhar como um grupo foi necessário fazer a base de dados acessível a todos, daí a necessidade de tornar a nossa base de dados online.
Por causa disso foi feita uma subscrição no Azure de forma a utilizar a sua ferramenta de conectar bases de dados MySQL.

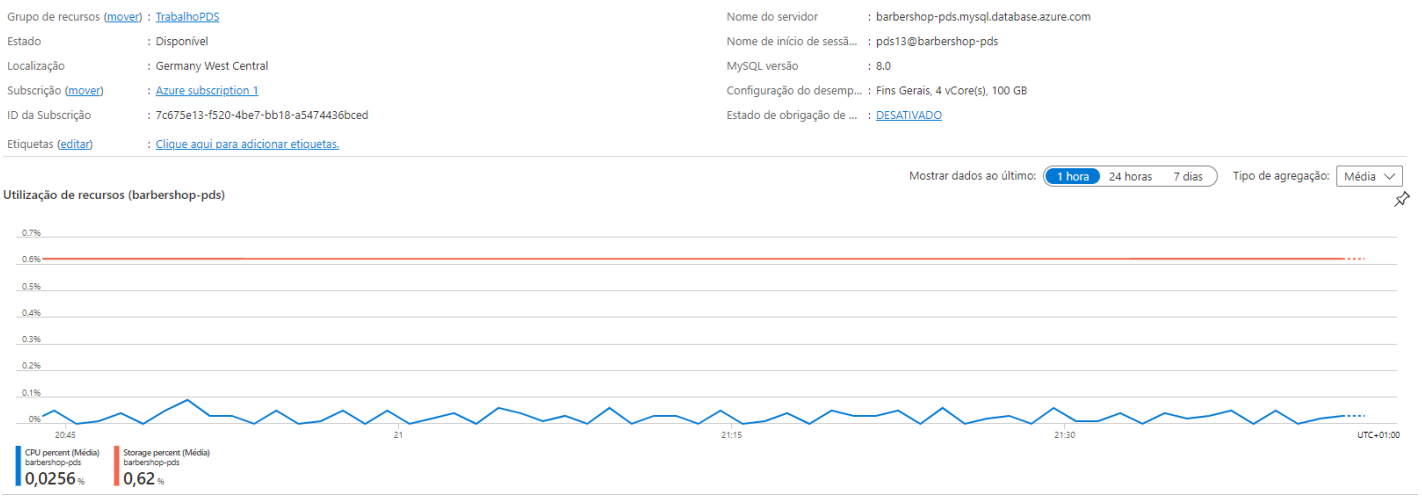


Gráfico de utilização da base de dados

Criação das tabelas

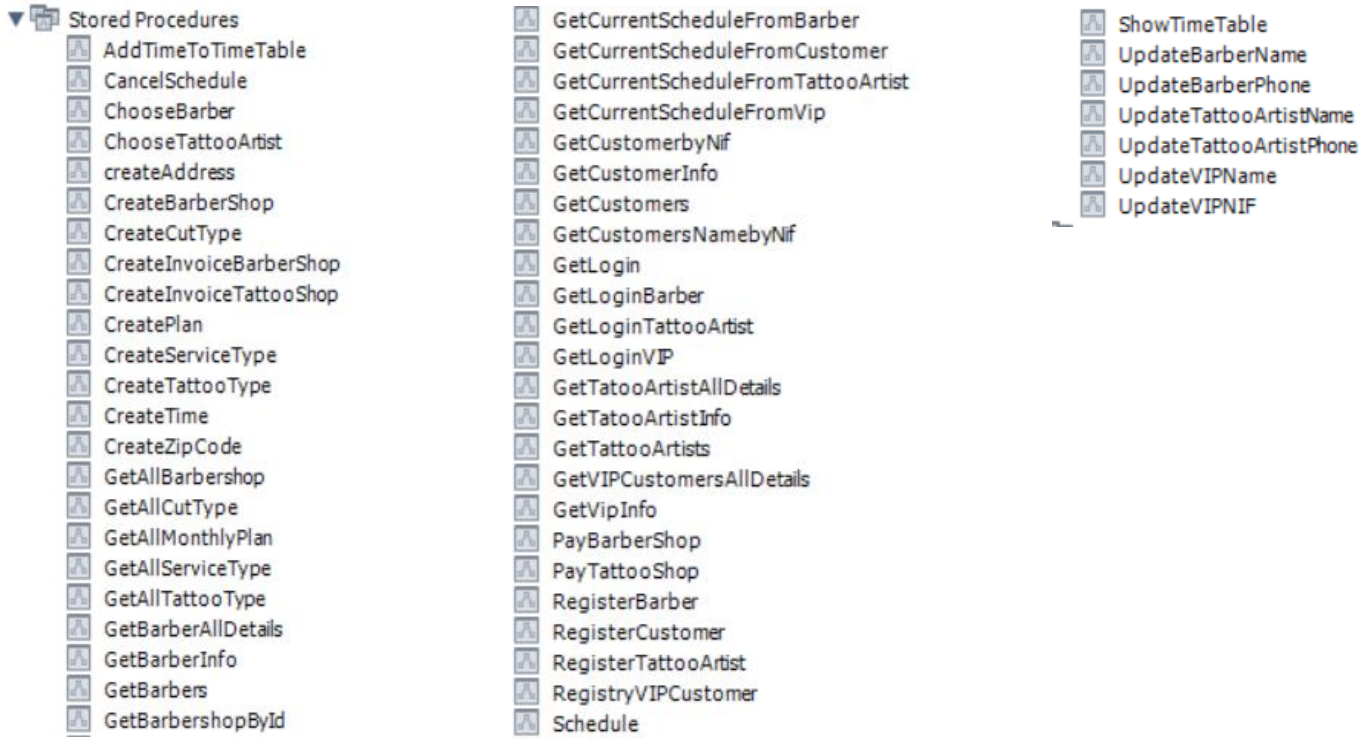
As tabelas da base de dados foram geradas pelo código do diagrama em cima apresentado.

```
1.  CREATE DATABASE  IF NOT EXISTS `barbershop` /*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE
    utf8mb4_0900_ai_ci */ /*!80016 DEFAULT ENCRYPTION='N' */;
2.  USE `barbershop`;
3.  CREATE TABLE BarberService (
4.    SCid        int(10) NOT NULL,
5.    Bid         int(10) NOT NULL,
6.    CutTypeCTId int(10) NOT NULL,
7.    PRIMARY KEY (SCid));
8.  CREATE TABLE Schedule (
9.    SCid            int(10) NOT NULL AUTO_INCREMENT,
10.   NIF             int(10) NOT NULL,
11.   scheduleDate    date,
12.   Bid             int(10) NOT NULL,
13.   ServiceTypeSTid int(10) NOT NULL,
14.   `Desc`          varchar(255),
15.   PRIMARY KEY (SCid));
16. CREATE TABLE Customer (
17.   NIF  int(10) NOT NULL,
18.   name varchar(255),
19.   PRIMARY KEY (NIF));
20. CREATE TABLE Barbershop (
21.   Bid  int(10) NOT NULL AUTO_INCREMENT,
22.   name varchar(255),
23.   Aid  int(10) NOT NULL,
24.   PRIMARY KEY (Bid));
25. CREATE TABLE Address (
26.   Aid    int(10) NOT NULL AUTO_INCREMENT,
27.   street varchar(255),
28.   number int(10),
29.   PCid   int(10) NOT NULL,
30.   PCext  int(10),
31.   PRIMARY KEY (Aid));
32. CREATE TABLE PostalCode (
33.   PCid     int(10) NOT NULL,
34.   Locality varchar(255),
35.   PRIMARY KEY (PCid));
36. CREATE TABLE ServiceType (
37.   STid   int(10) NOT NULL AUTO_INCREMENT,
38.   `Desc` varchar(255),
39.   PRIMARY KEY (STid));
40. CREATE TABLE Barber (
41.   LoginDataLid int(10) NOT NULL,
42.   Bid          int(10) NOT NULL AUTO_INCREMENT,
43.   name         varchar(255),
44.   phone        varchar(255),
45.   PRIMARY KEY (Bid));
46. CREATE TABLE LoginUser (
47.   Lid          int(10) NOT NULL AUTO_INCREMENT,
48.   email        varchar(255),
49.   password     varchar(255),
50.   PhotoURL     varchar(255),
51.   UserTypeTTId int(10) NOT NULL,
52.   PRIMARY KEY (Lid));
```

Excerto do código que gera todas as tabelas da base de dados

Stored Procedures

Conforme as necessidades do product backlog, foi necessário fazer dezenas de operações na base de dados de modo a poder adicionar, remover, alterar e pesquisar um determinado dado.
Nesse sentido foi então necessário criar uma stored procedure que atendesse cada necessidade do grupo.



Lista de todas as stored procedures criadas

Estas stored procedures podem fazer diversas coisas desde listar os barbeiros como adicionar um, entre outras coisas.

```
CREATE DEFINER=`pds13`@`%` PROCEDURE `GetBarberAllDetails`()
BEGIN
    DECLARE EXIT HANDLER FOR SQLException
    BEGIN
        SELECT 'Something was wrong';
        ROLLBACK;
    END;
    START TRANSACTION;
    SELECT loginuser.Lid as ID,barber.name,barber.phone,loginuser.email, barber.Bid FROM loginuser
        join  barber on loginuser.Lid = barber.LoginDataLid;
    COMMIT;
END
```

Stored Procedure que chama todos os barbeiros

Inicio da criação da API

Para a criação da API o grupo decidiu programa com JavaScript com o auxílio de uma ferramenta chamada Node.js.

Node.js é um ambiente de execução JavaScript que permite executar aplicações desenvolvidas com a linguagem de forma autónoma, sem depender de um navegador. Com ele, é possível criar praticamente qualquer tipo de aplicações web, desde servidores para sites estáticos e dinâmicos, até APIs e sistemas baseados em microserviços.

Além disso é uma ferramenta bastante intuitiva que incentiva a pesquisa e estudo próprio do desenvolvedor, pois além das suas bibliotecas aceita outras bibliotecas que permitem fazer inúmeras coisas apenas usando o mesmo ambiente de execução.

Para criar a aplicação web com os princípios do node foi usada outra ferramenta denominada Express. Além disso o Express provede vários frameworks comuns neste tipo de projetos.

Conexão à base de dados

De forma a poder fazer o projeto usando os dados da nossa base de dados é necessário criar uma ligação a ela, através do MySQL com o Node.js.

```
const connection = mysql.createPool({
    connectionLimit : 10,
    password: 'Trabalho1!',
    user: 'pds13@barbershop-pds',
    database: 'barbershop',
    host: 'barbershop-pds.mysql.database.azure.com',
    port: '3306'
});
```

Função responsável pela ligação à base de dados

Feita esta ligação é necessário atribuir as stored procedures criadas anteriormente às funções criadas no back end.
Por exemplo, se queremos criar uma função registe um cliente precisamos, de atribuir a essa função a stored procedure criada no MySQL encarregue de fazer isso.
Para ficar mais claro, olhemos para o exemplo abaixo.

```
static async RegisterCustomer(name,nif, result) {
    const[rows,fields] = await sql.execute(`CALL RegisterCustomer(${nif},"${name}")`);
    if (rows[0]) {
        //console.log("error: ", rows[0][0]);
        result("User exist", null);
    }else{
            //console.log("created customer");
            result(null,null);
        }
}
```

Função que chama a stored procedure "RegisterCustomer()"

Isto é feito para todas as funções que se encontram na pasta "db".

Controladores

Feita esta magia é necessário criar controladores para as funções criadas na pasta “db”.
Controladores são objetos JavaScript cujos métodos são responsáveis por tratar os dados de uma requisição HTTP, consultar ou alterar os dados necessários e retornar um resultado por meio de uma resposta adequadamente formatada.
Nesse sentido será possível exportar estas operações para os pedidos HTTP de forma a poder usar corretamente a API.

```
exports.GeAllBarberShops_get = async (req, res, next) => {
  try {
    await BarberShop.GeAllBarberShops((err, data) => {
      if (err)
        res.status(500).json({
          message:
            err.message || "Ocorreu algum erro para obter as barbearias"
        });
      else res.status(200).json(data);
    });
  } catch (error) {
    console.log(error);
    next(error);
  }
};
```

Controlador de GetALLBarberShops

O objeto “req” representa o pedido HTTP e tem propriedades para o pedido de cadeia de consulta, parâmetros, corpo e cabeçalhos HTTP. O objeto “res” representa a resposta HTTP que uma aplicação Express envia quando recebe um pedido HTTP.

O estado “500” significa que algo inesperado correu mal.
O estado “200” significa que tudo correu como o esperado.

Rotas

Uma rota dentro de uma API é um caminho específico a tomar para obter informações ou dados específicos.
Para isto ser possível necessitamos de importar as bibliotecas e as funcionalidades do Express referido em cima.

```
var router = require("express").Router();
```

Criação de uma variável router com as propriedades do Express

Os métodos HTTP mais comuns são:

- POST: Usado para criar nova informação
- GET: Usado para recolher informação
- PUT: Usado para atualizar informação
- DELETE: Usado para eliminar informação

Usando o exemplo do ficheiro “login_router.js” podemos verificar que são usados quatro métodos POST para poder criar novos logins.

```
router.post("/barber",Login.GetLoginBarber_POST);
router.post("/tattoartist",Login.GetLoginTattooArtist_POST);
router.post("/vip",Login.GetLoginVIP_POST);
router.post("/common",Login.GetLoginNormalCustomer_POST);
```

Posts destinados a fazer os logins

Todos estes POSTs vão levar-nos a páginas diferentes indicadas neste métodos.

Frontend

Para o desenvolvimento do frontend da aplicação decidimos usar Angular.
Angular é uma plataforma e framework para construção da interface de aplicações usando HTML, CSS e, principalmente, JavaScript, criada pelos desenvolvedores da Google.
Além disso a plataforma merece destaque no fato de que ela é open source, possui uma grande comunidade, existem várias empresas utilizando e tem muito material de estudo para quem deseja se aperfeiçoar.
Em suma foi bastante simples e lúcida de se utilizar.
Claro que para o Angular funcionar foi necessário usar as bases de HMTL e CSS.

Todas as funções criadas no backend são chamadas para o frontend através do ficheiro “api-service.service.ts”.

```
getAllbarbers(){
    return this._http.get(this.barberUrl+"/all");
}

getAllTatooArtist(){
    return this._http.get(this.tatooartistUrl+"/all");
}
```

Algumas das funções localizadas no “api-service.service.ts”

Além disso neste mesmo ficheiro é feito um Cliente HTTP.
HttpClient é uma classe de serviço incorporada disponível no pacote @angular/comum/http.
Tem vários tipos de assinatura e devolução para cada pedido.
Utiliza as APIs de base observável RxJS, o que significa que devolve o observável e o que precisamos para subscrevê-lo.
Esta API foi desenvolvida com base na interface XMLHttpRequest exposta pelos navegadores.
É necessário o uso deste serviço, se queremos passar as nossas coisas online.

```
constructor(private _http: HttpClient) { }

apiUrl = `http://localhost:3000`
barberUrl = `${this.apiUrl}/barber`;
tatooartistUrl = `${this.apiUrl}/tatooartist`;
customerUrl = `${this.apiUrl}/customer`;
vipcustomerUrl = `${this.apiUrl}/vipcustomer`;
choiceUrl = `${this.apiUrl}/choice`;
loginUrl= `${this.apiUrl}/login`;
timetableUrl= `${this.apiUrl}/timetable`;
planUrl= `${this.apiUrl}/plan`;
servicetypeUrl = `${this.apiUrl}/servicetype`;
cuttypeUrl= `${this.apiUrl}/cuttype`;
tatootypeUrl= `${this.apiUrl}/tatootype`;
barbershopUrl= `${this.apiUrl}/barbershop`;
```

Cliente HTTP e todos os URLs necessários na aplicação

Depois disto tudo, no mesmo ficheiro são exportadas todas as interfaces que vão ser chamadas no "module.ts" de todos os componentes.

```
export interface EmployeeRegisterData {
  email?:any;
  phone?:any;
  pass?:any;
  name?:any;
}
```

```
signupBarber : EmployeeRegisterData = {
  email: '',
  phone: '',
  pass: '',
  name:'',

}
```

Interface exportada no "api-service.service.ts" e a sua respetiva chamada no "module.ts" do componente "signup-barber"

Componentes

Cada página feita vai ser dividida em vários componentes o que possibilita o usso de componentes já criados noutras páginas.
Por exemplo na nossa página inicial temos o seu componente principal chamado “app-home-page”.

```
@Component({
  selector: 'app-home-page',
```

Componente “app-home-page”

Mas depois no HTML dessa página vamos chamar outros componentes respetivos a certas partes da página.

```
<div class="hero"><app-nav></app-nav></div>
<div class="content"><app-content></app-content></div>
<section class="about"><app-about></app-about></section>
<footer><app-footer><app-social></app-social></app-footer></footer>
```

Diversas componentes no HTML da página inicial

Por exemplo o “<footer>” referido em cima é uma componente independente que só precisa de ser chamada.

```
<footer>
<p>Contacto</p>
<p>Não hesites em contactar com qualquer duvida</p>
<app-social></app-social>
<p class="end">Desde 2022</p>
<div class="hankstudiosgmailcom">hankstudios@gmail.com</div>
</footer>
```

Componente “app-footer”

Esta lógica é aplicada ao longo de todo o projeto, o que permite uma maior flexibilidade no desenvolvimento do frontend da aplicação.

Terminal

Como a nossa aplicação contém um frontend e um backend em pastas diferentes necessitamos de utilizar dois terminais um para poder correr o backend usando o comando **“npm run dev”** e outro para correr o frontend usando o comando **“npm start”.**

```
PS C:\Users\André Filipe\OneDrive\Ambiente de Trabalho\Wiki\wiki\Angular-Pw\BarberShop_Backend\backend> npm run dev

> barber_shop_api@1.0.0 dev
> npx nodemon server/server.js -w server

[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): server\**\*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server/server.js`
Server is running on port: 3000
Server is running on url: http://localhost:3000
```

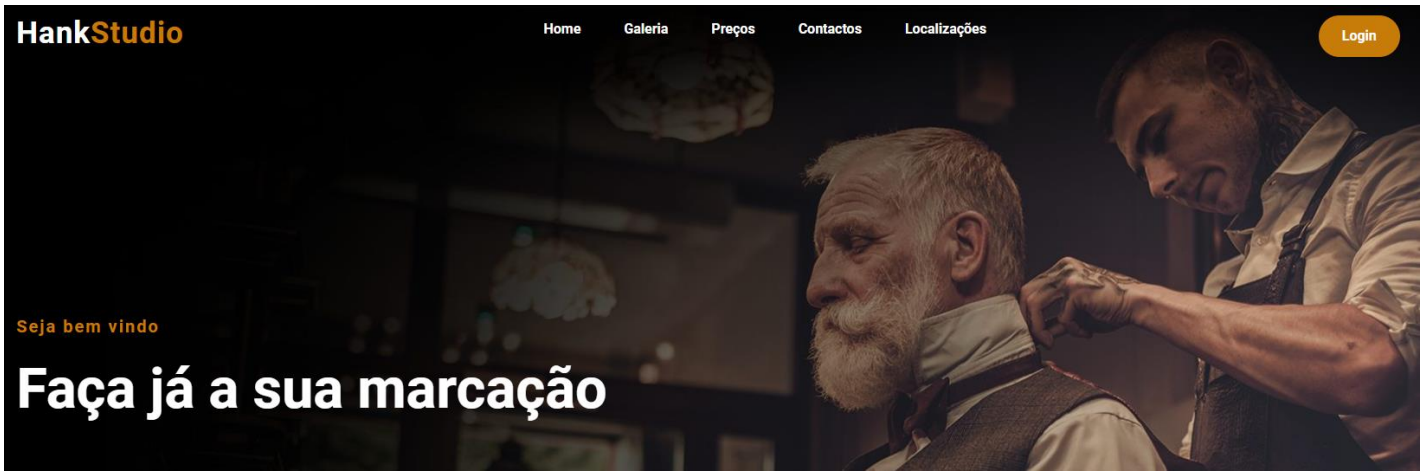
Terminal usado para o backend

```
PS C:\Users\André Filipe\OneDrive\Ambiente de Trabalho\Wiki\wiki\Angular-Pw\BarberShop> npm start

> barber-shop@0.0.0 start
> ng serve

Node.js version v17.9.0 detected.
Odd numbered Node.js versions will not enter LTS status and should not be used for production. For more information, please see https://nodejs.org/en/about/releases/.
✓ Browser application bundle generation complete.

Initial Chunk Files   | Names         |  Raw Size
vendor.js             | vendor        |   2.34 MB |
main.js               | main          | 540.88 kB |
polyfills.js          | polyfills     | 294.84 kB |
styles.css, styles.js | styles        | 174.01 kB |
runtime.js            | runtime       |   6.52 kB |

                      | Initial Total |   3.33 MB

Build at: 2022-06-08T22:36:24.881Z - Hash: 9612d62a37e6f327 - Time: 5899ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
```

Terminal usado para o frontend

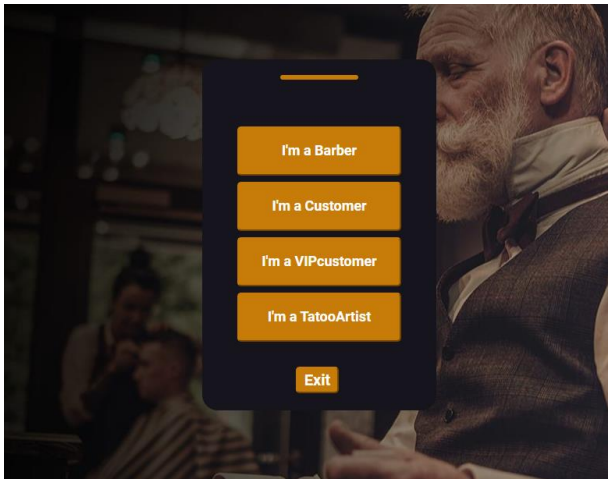
Site

Feito isto podemos finalmente rodar o site criado.
Iremos ter uma página inicial, uma página de logins, uma galeria entre outras coisas.

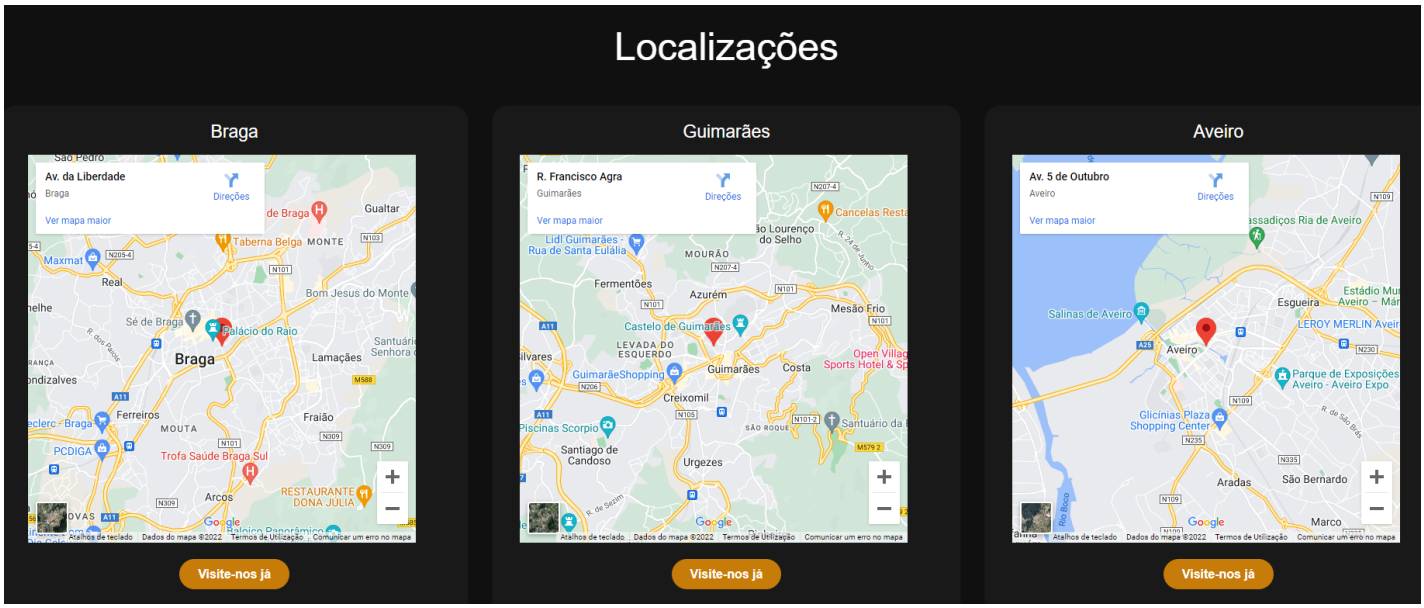


Excerto da página inicial

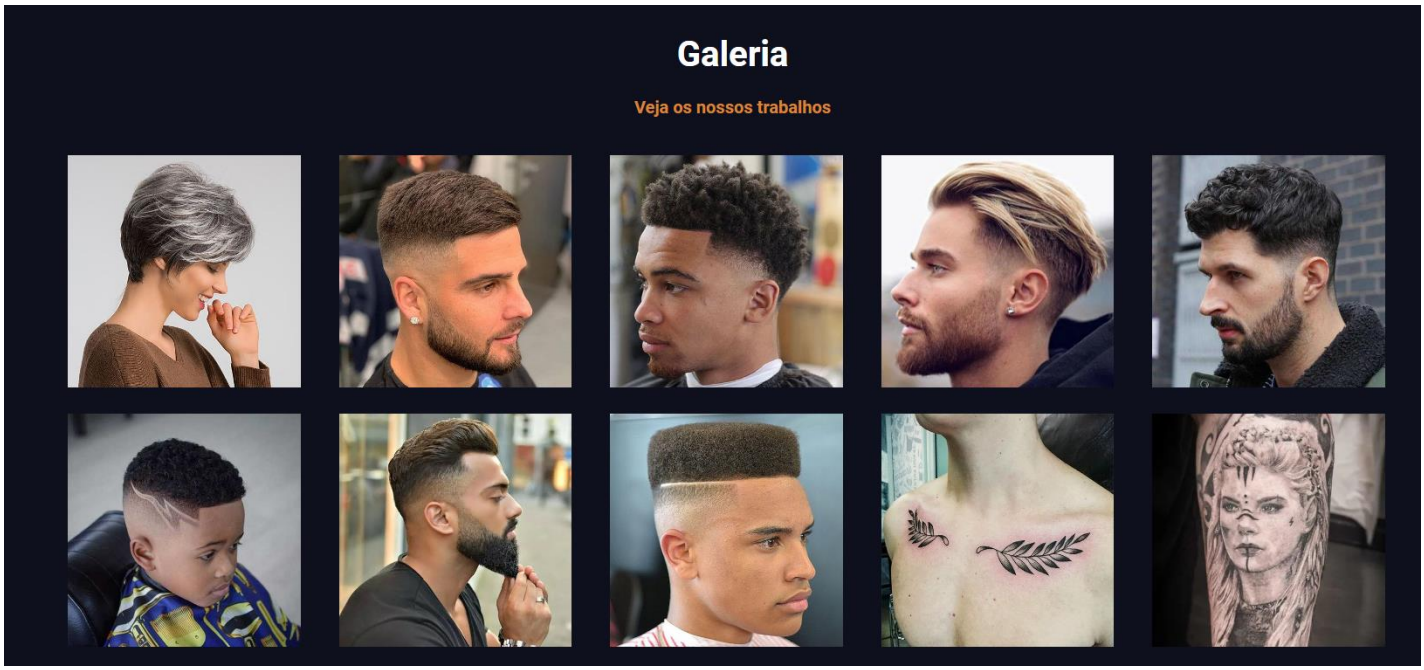
Footer referido em cima



Excerto da página de escolha



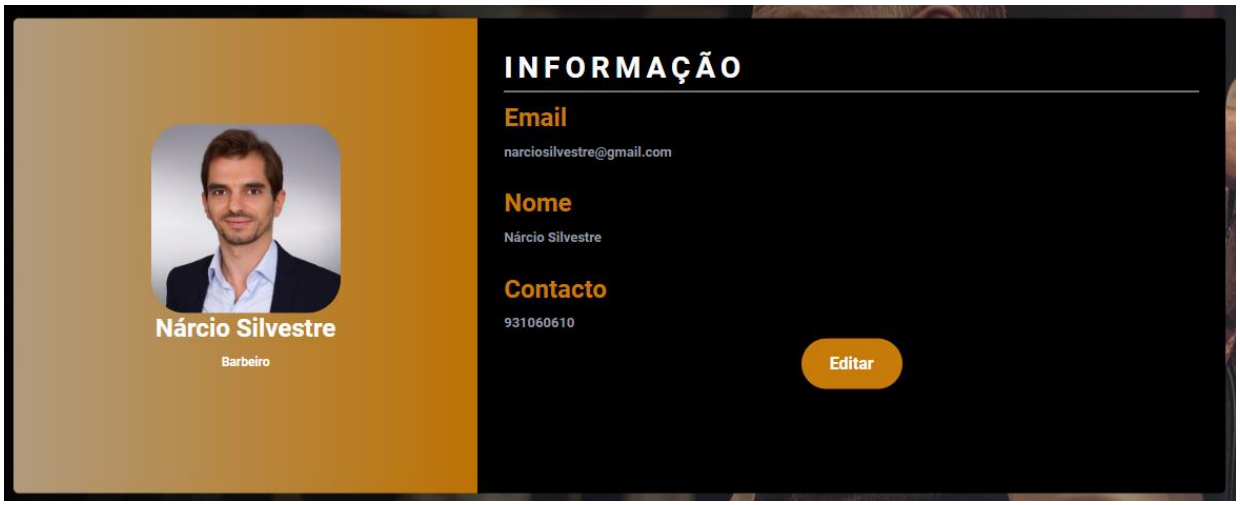
Página que demonstra a localização das barbearias no mapa



Galeria que dispõem de várias imagens de referência seja de cortes de cabelo ou tatuagens

Área de funcionário do barbeiro Nárcio Silvestre

As suas respetivas marcações



A sua página de perfil

Bibliografia

Plataforma Moodle: https://elearning1.ipca.pt/

Visual Paradigm: https://www.visual-paradigm.com

Projeto do Jira Software:
https://a21126.atlassian.net/jira/software/projects/BMS/boards/2

Figma: https://www.figma.com/

Repositório GitHub: https://github.com/syke007/Angular-Pw

ATA de reuniões:
https://docs.google.com/document/d/1lkZyAMQRWB15LUnXBX93TnYlHTlBEC_sIoWqKN_CkK0/edit?usp=sharing

Documentação do código:
https://app.swaggerhub.com/apis/justAndre02/BarberShop/1.0.0