



MESTRADO EM INFORMÁTICA MÉDICA

APLICAÇÕES DISTRIBUÍDAS

---

## CuidaUM - Registo de dados e cuidados de saúde

---

**Realizado por:**

Beatriz Rodrigues, PG53696

Bruno Machado, PG53709

Rúben Ganança, PG54203

**Docente:** António Luís Pinto Ferreira Sousa

17 de janeiro de 2024

# Índice

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Preliminares</b>	<b>4</b>
2.1	<i>Django</i> . . . . .	4
2.2	Arquitetura Cliente-Servidor . . . . .	5
<b>3</b>	<b>Descrição do Trabalho e Análise de Resultados</b>	<b>6</b>
3.1	<i>Models</i> . . . . .	6
3.1.1	Utilizador e semelhantes . . . . .	6
3.1.2	Medicamento . . . . .	8
3.1.3	Consulta . . . . .	8
3.1.4	Prescricao, Exame, Parametros . . . . .	9
3.2	Relacionamento entre Entidades . . . . .	10
3.3	<i>Forms</i> . . . . .	10
3.3.1	<i>UtenteCreationForms, MedicoCreationForms, FamiliarCre-</i> <i>ationForms</i> . . . . .	10
3.3.2	<i>UtenteLoginForm, MedicoLoginForm, FamiliarLoginForm</i> .	11
3.3.3	<i>PrescricaoCreationForm, ConsultaCreationForm, Exame-</i> <i>CreationForm, ParametrosCreationForm</i> . . . . .	11
3.4	<i>Views</i> . . . . .	12
3.4.1	Listar . . . . .	13
3.4.2	Criar . . . . .	14
3.4.3	Detalhes . . . . .	14
3.4.4	<i>Home, home_utente, home_medico e home_familiar</i> . . .	15
3.4.5	Registar e <i>Login</i> . . . . .	16
3.5	<i>Templates</i> . . . . .	17
3.5.1	<i>Template home.html</i> . . . . .	17
3.5.2	<i>Template registar.html</i> . . . . .	18
3.5.3	<i>Template login.html</i> . . . . .	19

---

3.5.4	<i>Templates home_utente.html, home_medico.html e home_familiar.html</i> . . . . .	19
3.5.5	<i>Template</i> relativos às funções Criar . . . . .	20
3.5.6	<i>Template</i> relativos às funções Listar . . . . .	21
3.5.7	<i>Template</i> relativos às funções Detalhes . . . . .	22
3.6	<i>URLs</i> . . . . .	23
3.7	<i>Site Admin</i> . . . . .	23
3.8	Fatores Valorização . . . . .	24
3.8.1	Estatística . . . . .	24
3.8.2	Registo e Autenticação de utilizadores . . . . .	25
<b>4</b>	<b>Conclusão</b>	<b>26</b>

## 1. Introdução

No âmbito da Unidade Curricular de Aplicações Distribuídas o trabalho seguinte foi realizado com a finalidade de implementar uma aplicação ou conjunto de aplicações que seja capaz de gerir e registar cuidados de saúde e dados fisiológicos de uma unidade de cuidados de saúde. Para tal foi usado a *framework* Django que utiliza uma arquitetura **Servidor-Cliente**.

Deste modo, o principal problema deste programa passou por conseguir implementar um **sistema multiutilizador** de gestão de dados, para o registo de entidades ou conseguir listá-as. É de se notar que cada entidade ou tipo de utilizador nesta aplicação pode ter vários tipos de atributos associados.

Os principais requisitos deste trabalho passam por:

- Criação de perfis para os utilizadores;
- Definição e implementação das entidades envolvidas;
- Implementação do processo de registo e *login* de entidades;
- Implementação do processo de registo de vários tipos de dados (Consultas, Prescrições, entre outros);
- Implementação do processo de consulta destes dados;

Assim, e de forma a cumprir o acima mencionado definiu-se as entidades tal como as funcionalidades envolvidas no projeto, tendo a conta a lógica idealizada para o correto funcionamento do programa.

## 2. *Preliminares*

### 2.1 *Django*

Um *framework*, tal como uma biblioteca, pode ser equiparado a uma caixa de ferramentas: um conjunto de recursos que auxiliam na construção de um projeto. O *Django* é, portanto, uma *framework* do tipo *web full stack open source* (código aberto) baseado na linguagem *Python*, que é uma linguagem gratuita de alto nível [1].

Este *framework* foi criado com o intuito de resolver problemas associados ao processo de desenvolvimento de aplicações *web* nomeadamente o processo de autenticação e registo de utilizadores, rotas, *object relational mapper* (ORM) e até *migrations* [2].

Uma das principais características, como já mencionado, é o ORM em que se define a modelagem dos dados através de classes. Através deste processo é possível gerar tabelas na base de dados sem a necessidade de as manipular através do *SQL*.

Outra vantagem proveniente do uso de uma *framework* como o *Django* é a capacidade de criar projetos escaláveis e que tem um sistema de segurança eficiente. De modo geral, um projeto desenvolvido com o *Django* costuma ser dividido em pequenas aplicações compostas por pacotes *Python* que, por sua vez, resolvem porções específicas de requisitos específicos da aplicação. Esta divisão do projeto segue um padrão denominado por **MVT** (*Model View Template*) em que cada projeto acompanhado por um conjunto de diretorias e ficheiros pré-definidos.

## 2.2 Arquitetura Cliente-Servidor

A arquitetura cliente-servidor é um modelo muito relevante para o desenvolvimento de sistemas distribuídos, onde o processamento das informações é dividido entre os dois elementos principais: o cliente, encarregado de solicitar serviços e o servidor, responsável por atender a essas solicitações. A arquitetura referida é amplamente utilizada, por exemplo, no desenvolvimento de *websites*, como é o caso.

Desta forma, o cliente inicia a interação enviando uma solicitação ao servidor, o qual, por sua vez, executa as tarefas solicitadas e retorna uma resposta ao cliente. Essa abordagem modular facilita a manutenção, atualização e escalabilidade dos sistemas, proporcionando uma clara separação de responsabilidades.

Como mencionado, uma das aplicações práticas deste modelo encontra-se no desenvolvimento de aplicações *web*, onde se utiliza um navegador com capacidade de apresentar o conteúdo das páginas *web* requisitadas, juntamente com um banco de dados e um servidor de aplicação, sendo o *Django* um exemplo específico desse tipo de servidor. O papel essencial do *Django* reside no processamento eficiente da informação, atuando como um componente fundamental para o funcionamento coeso dessas aplicações *web*.

## 3. *Descrição do Trabalho e Análise de Resultados*

### 3.1 *Models*

É necessário registar diversos tipos de dados, para tal, desenvolveram-se modelos para cada entidade envolvida e as múltiplas interações entre as mesmas (nomeadamente, *Utilizador*, *Medico*, *Utente*, *Familiar*, *Medicamento*, *Exame*, *Prescricao*, *Parametros*, *Consulta*)

Os modelos criados servem como fontes definitivas e exclusivas das informações, definindo os campos essenciais e comportamentos associados aos dados, proporcionando uma estrutura organizada e eficiente para a gestão dos mesmos em aplicações distribuídas [3].

#### 3.1.1 *Utilizador e semelhantes*

No desenvolvimento da aplicação, criou-se o modelo Utilizador para representar a entidade "Utilizador", que depois irá ramificar-se noutras 3 entidades: Utente, Medico e Familiar, retratando os 3 tipos de utilizador que a aplicação terá.

Este modelo está representado na Figura 3.1 e possui os atributos associados: nome, sexo, número de saúde, tipo, que são todos do tipo *CharField*, com comprimentos máximos diferentes dependendo da necessidade de cada um. Ainda existem outros atributos de outros tipos como a idade que é do tipo *IntegerField* e o email que é um *EmailField*.

Relativamente ao ID de cada utilizador, o mesmo é criado de forma automática e incrementada pelo sistema de gestão da base de dados sempre que uma nova instância é criada.

```

class Utilizador(AbstractUser):
    TIPO_CHOICES = [
        ('M', 'Médico'),
        ('U', 'Utente'),
        ('F', 'Familiar'),
    ]

    nome = models.CharField(max_length=200, unique=False)
    idade = models.IntegerField(default=18)
    sexo = models.CharField(max_length=10, choices=[("M", "Masculino"), ("F", "Feminino")])
    email = models.EmailField(max_length=50, unique=True)
    numSaude = models.CharField(max_length=9, unique=True, validators=[MinLengthValidator(limit_value=9), MaxLengthValidator(limit_value=9)])
    tipo = models.CharField(max_length=1, choices=TIPO_CHOICES)

    USERNAME_FIELD = "email"
    REQUIRED_FIELDS = ["username"]

    groups = models.ManyToManyField(Group, related_name='utilizador_groups')
    user_permissions = models.ManyToManyField(Permission, related_name='utilizador_user_permissions')

```

Figura 3.1: Modelo associado aos utilizadores

As entidades Utente, Medico e Familiar herdam os atributos da classe Utilizador e ainda acrescentam os seus próprios atributos únicos. No caso do médico é importante estar estabelecida a informação relativa à sua especialidade e no familiar informação relativa ao utente a que está associado e qual o seu grau de parentesco com o mesmo. Estas entidades podem ser observadas na figura 3.2.

```

class Medico(Utilizador):
    TIPO_ESPECIALIDADES = [
        ("C", "Cardiologia"),
        ("P", "Pediatria"),
        ("Ps", "Psiquiatria"),
        ("CG", "Clínica Geral"),
        ("N", "Neurologia"),
    ]
    especialidade = models.CharField(max_length=2, choices= TIPO_ESPECIALIDADES)

class Utente(Utilizador):
    pass

class Familiar(Utilizador):
    GRAU_PARENTESCO = [
        ("I", "Irmã/Irmão"),
        ("P", "Pai"),
        ("M", "Mãe"),
        ("O", "Outro")
    ]

    relacao = models.CharField(max_length=2, choices = GRAU_PARENTESCO)
    id_associado = models.ForeignKey(Utente, on_delete=models.CASCADE, related_name='familiares')

    def get_utente_url(self):
        return reverse('detalhes_utente', args=[str(self.id_associado.pk)])

```

Figura 3.2: Modelo associado aos medicos, utentes e familiares



### 3.1.2 Medicamento

De forma a que diferentes medicamentos possam ser prescritos numa prescrição foi necessário criar uma entidade que guardasse informação relativa aos mesmos. Deste modo, esta entidade apenas consta com informação relativamente ao nome do mesmo, informação essa que é do tipo *CharField*.

```
class Medicamento(models.Model):
    nome = models.CharField(max_length=200, unique=True)

    def __str__(self):
        return self.nome
```

Figura 3.3: Modelo associado aos medicamentos

### 3.1.3 Consulta

De forma a ser possível marcar consultas foi necessário criar uma entidade dedicada a este efeito. Desse modo, cada consulta tem a si associado um utente e médico através de *ForeignKey* com as entidades *Utente* e *Medico*, respetivamente. Esta relação permite que cada consulta tenha associada a si apenas um utente e um médico, mas que cada médico ou utente possa ter várias consultas.

Para além de uma consulta ter um utente e médico associado, ela pode ter várias prescrições, exames e parâmetros associados também. O último atributo que consta nesta entidade é a data e hora em que a consulta ficou marcada. Esta entidade encontra-se na figura 3.4.

```
class Consulta(models.Model):
    utente = models.ForeignKey(Utente, on_delete=models.CASCADE)
    medico = models.ForeignKey(Medico, on_delete=models.CASCADE)
    prescricoes = models.ManyToManyField(Prescricao, related_name='consultas_prescritas', null=True, blank=True)
    exames = models.ManyToManyField(Exame, related_name='exames_prescritos', null=True, blank=True)
    parametros = models.ManyToManyField(Parametros, related_name='parametros_prescritos', null=True, blank=True)
    data = models.DateTimeField(default=timezone.now)

    def __str__(self):
        return f"{self.data}, {self.utente.nome}"
```

Figura 3.4: Modelo associado às consultas

### 3.1.4 Prescricao, Exame, Parametros

Para a correta assimilação das informações associadas às prescrições, exames e parâmetros físicos de um utente, criaram-se modelos para cada uma destas informações.

Deste modo, as entidades Exame e Parametros, para além dos seus atributos próprios, têm uma *ForeignKey* com o Utente e a Consulta, de modo a conseguirem manter uma relação entre os mesmos e associar um parametro e exame a cada consulta. Por outro lado, a Prescricao tem também uma *ForeignKey* associada ao Utente e Consulta, como tem também a si associada informação relativamente aos medicamentos, de modo a conseguir receitar um medicamento da lista existente.

As entidades Prescricao e Exame podem ser observadas na figura 3.5

```
class Prescricao(models.Model):
    TOMAS = [
        ("U", "Uma vez ao dia"),
        ("D", "Duas vezes ao dia"),
        ("T", "Três vezes ao dia"),
        ("Q", "Quatro vezes ao dia"),
        ("O", "Outros"),
    ]

    utente = models.ForeignKey(Utente, on_delete=models.CASCADE)
    medicamento = models.ForeignKey(Medicamento, on_delete=models.CASCADE)
    data_prescricao = models.DateField(default=timezone.now)
    data_termino = models.DateField()
    tomas = models.CharField(max_length=1, choices= TOMAS)
    consulta = models.ForeignKey('Consulta', on_delete=models.CASCADE)

class Exame(models.Model):
    TIPO_EXAMES = [
        ("E", "ECG"),
        ("R", "RaiosX"),
        ("H", "Hemograma"),
        ("M", "Ressonância Magnética"),
        ("C", "Ecografia"),
        ("O", "Outros"),
    ]

    utente = models.ForeignKey(Utente, on_delete=models.CASCADE)
    tipo_exam = models.CharField(max_length=1, choices= TIPO_EXAMES)
    consulta = models.ForeignKey('Consulta', on_delete=models.CASCADE)
    data = models.DateField(default=timezone.now)
```

Figura 3.5: Modelo associado às prescrições e exames

## 3.2 Relacionamento entre Entidades

As relações entre as diferentes entidades mencionadas na secção anterior são necessárias ter em conta ao longo da criação de uma aplicação. Para melhor evidenciar na figura 3.6 estão apresentadas as entidades e a forma como se relacionam entre si, para uma melhor percepção.

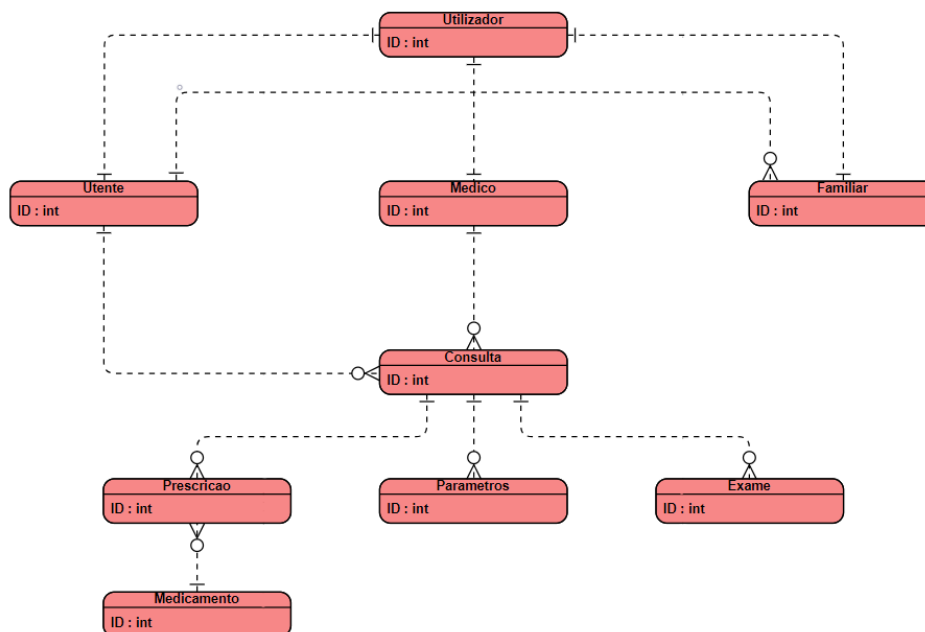


Figura 3.6: Relacionamento entre as Entidades existentes

## 3.3 Forms

O desenvolvimento de *forms* é fulcral, uma vez que permite passar de um domínio de um servidor que é apenas capaz de publicar conteúdo para um domínio de um servidor aplicacional, que já é capaz de publicar conteúdo e de receber inputs dos diferentes utilizadores e de responder aos mesmos.

### 3.3.1 *UtenteCreationForms*, *MedicoCreationForms*, *FamiliarCreationForms*

Estes *forms* têm o propósito de registar os diferentes tipos de utilizador. Neste formulário pretende-se que todas as informações sejam preenchidas pelo utilizador aquando o seu registo. Um exemplo destes forms encontra-se na figura 3.7

(os *MedicoCreationForms* e *FamiliarCreationForms* são semelhantes ao exemplo ilustrado).

```
class UtenteCreationForms(UserCreationForm):
    class Meta(UserCreationForm.Meta):
        model = Utente
        fields = ["nome", "idade", "sexo", "tipo", "numSaude", "email", "password1", "password2"]

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        # Definir o valor do campo 'tipo' como 'U' ao inicializar o formulário

        self.fields["tipo"].initial = "U"
        self.fields["tipo"].widget = forms.HiddenInput()
        self.fields["tipo"].label = ""
        self.fields["numSaude"].label = "Número de Saúde"
```

Figura 3.7: *Forms* associado ao registo de utentes

### 3.3.2 *UtenteLoginForm, MedicoLoginForm, FamiliarLoginForm*

Para que os utilizadores se possam registar foi necessário criar um *forms* para tal. Deste modo, criou-se um *forms* para cada tipo de utilizador de forma a diferenciar o tipo de utilizador que está a entrar na sua conta. De forma a fazer um *login* mais intuitivo o seu login é feito pelo *email*.

Um exemplo deste tipo de *forms* pode ser visualizado na figura 3.8.

```
class UtenteLoginForm(forms.Form):
    email = forms.EmailField(label="Email")
    password = forms.CharField(label="Password", strip=False, widget=forms.PasswordInput)
```

Figura 3.8: *Forms* associado ao login de utentes

### 3.3.3 *PrescricaoCreationForm, ConsultaCreationForm, ExameCreationForm, ParametrosCreationForm*

Estes formulários têm como objetivo marcar/registar prescrições médicas, exames, parâmetros físicos ou consultas. É importante referir que como as prescrições, exames e parâmetros estão associados a uma consulta eles só podem ser associados a uma consulta já marcada. Note-se também que estes *forms* só podem ser acedidos por utilizadores que são médicos, dado que apenas médicos têm

permissões para fazer estas operações.

Como exemplo, encontra-se na figura 3.9 um destes *forms*, sendo que os restantes seguem uma estrutura muito similar à demonstrada.

```
class PrescricaoCreationForm(forms.ModelForm):  
    class Meta:  
        model = Prescricao  
        fields = ["utente", "medicamento", "data_prescricao", "data_termino", "tomas", "consulta"]  
  
    def __init__(self, user, *args, **kwargs):  
        super().__init__(*args, **kwargs)  
  
        # Se o usuário for um médico, mostrar apenas os utentes associados a ele  
        if user.tipo == 'M':  
            self.fields["utente"].queryset = Utente.objects.filter(consulta__medico=user).distinct()  
            self.fields["consulta"].queryset = Consulta.objects.filter(medico=user)  
        else:  
            self.fields["utente"].queryset = Utente.objects.all()  
  
        self.fields["utente"].queryset = Utente.objects.all()  
        self.fields["medicamento"].queryset = Medicamento.objects.all()  
        self.fields["data_prescricao"].label = "Data da prescrição"  
        self.fields["data_prescricao"].initial = timezone.now()  
        self.fields["data_termino"].label = "Data de término"  
        self.fields["data_termino"].initial = timezone.now()
```

Figura 3.9: *Forms* associado ao registo de prescrições

## 3.4 Views

Uma *view* é uma função em *Python* que recebe um *request* (pedido) *web* e que retorna uma resposta que, no caso do trabalho em questão, baseou-se na resposta de uma página *web HTML*. Ou seja, há redirecionamento de uma página para outra, de acordo com o *url* a que se encontra associada, um exemplo desta situação é aquando o preenchimento de um formulário de registo de utilizador há o redirecionamento para o seu *login*. Uma *view* é, portanto, a ponta de ligação entre as diferentes secções dos *urls* e das páginas *HTML* [4].

No contexto deste trabalho, grande parte das *views* necessitam que o login do utilizador esteja feito, de forma a retornarem uma resposta. Isto acontece, uma vez que diferentes tipos de utilizador têm acesso a diferentes tipos de operações, sendo portanto necessário restrições para que, por exemplo, utentes não sejam capazes de criar uma consulta.

### 3.4.1 Listar

Estas funções têm o objetivo de retornar todo o tipo de informações relativas a uma entidade. Existem funções para listar todos os utentes, médicos, familiares e medicamentos, que são apenas acessíveis por utilizador que sejam médicos e há também funções de listagem que são acessíveis por todo o tipo de utilizadores, como a lista de consultas, exames, prescrições e parâmetros. Note-se que estas últimas são acessíveis por todos os tipos de utilizador, mas os utentes só podem ver as informações relativas a si mesmos e o familiar ao utente que está associado, enquanto que o médico pode aceder a toda a informação.

De notar que na função de listagem das consultas é realizada uma separação das consultas passadas e das futuras, tendo em consideração a data atual, de modo a facilitar a visualização das mesmas, para tal, faz-se uso de um filtro.

Um exemplo de uma função para listar familiares e de uma função para listar consultas, encontra-se na figura 3.10:

```
@user_passes_test(is_medico, login_url='login')
def listar_familiares(request):
    familiares = Familiar.objects.all()
    return render(request, 'listar_familiares.html', {'familiares': familiares})

@login_required(login_url="login")
def listar_consultas(request):
    user = request.user

    if user.is_authenticated:
        now = timezone.now().date()

        if user.tipo == 'U':
            consultas_passadas = Consulta.objects.filter(utente=user.utente, data__date__lt=now)
            consultas_futuras = Consulta.objects.filter(utente=user.utente, data__date__gt=now)
        elif user.tipo == 'M':
            consultas_passadas = Consulta.objects.filter(medico=user.medico, data__date__lt=now)
            consultas_futuras = Consulta.objects.filter(medico=user.medico, data__date__gt=now)
        elif user.tipo == "F":
            consultas_passadas = Consulta.objects.filter(utente = user.familiar.id_associado, data__date__lt=now)
            consultas_futuras = Consulta.objects.filter(utente = user.familiar.id_associado, data__date__gt=now)
        else:
            # Adicione um comportamento padrão ou retorne listas vazias para outros tipos de usuários
            consultas_passadas = []
            consultas_futuras = []

    return render(request, 'listar_consultas.html', {'consultas_passadas': consultas_passadas, 'consultas_futuras': consultas_futuras})
else:
    # Caso o usuário não esteja autenticado, você pode redirecioná-lo para a página de login
    return redirect('listar_utentes')
```

Figura 3.10: Funções associadas à listagem de familiares e consultas

### 3.4.2 Criar

Estas funções têm o objetivo de criar novas consultas, prescrições, exames e parâmetros com todas as informações necessárias para tal. Assim, para, por exemplo, ser criada uma consulta faz-se uso do *ConsultaCreationForm* de forma a receber como input todos os atributos da consulta a adicionar.

É relevante referir que estas funções apenas podem ser acedidas por médicos e se as operações forem concluídas com sucesso o médico é redirecionado para a *home\_medico*.

De notar que tanto as prescrições, como exames e parâmetros, estão associados à própria consulta onde forem criados.

Um exemplo de uma função para criar parâmetros, encontra-se na figura 3.11:

```
def criar_parametro(request):  
    if request.method == "POST":  
        user = request.user  
        form = ParametrosCreationForm(user, request.POST)  
        if form.is_valid():  
            parametro = form.save()  
  
            consulta_associada = parametro.consulta  
  
            consulta_associada.parametros.add(parametro)  
  
            return redirect('home_medico')  
    else:  
        user = request.user  
        form = ParametrosCreationForm(user)  
        medico_id = request.user.id  
    return render(request, 'criar_parametro.html', {'form': form, "medico_id": medico_id})
```

Figura 3.11: Função associada à criação de parâmetros

### 3.4.3 Detalhes

Estas funções foram criadas com o intuito de fornecer os detalhes relativos aos utentes, familiares, consultas, prescrições, exames, parâmetros e médicos, retornando as informações relativas a cada um.

Existem algumas restrições relativamente ao acesso dos detalhes. Os detalhes

do médico apenas podem ser acedidos pelo próprio médico. O médico consegue aceder a todos os detalhes associados aos seus utentes. O utente apenas consegue visualizar os detalhes associados a si mesmo (as suas informações, consultas, prescrições, parâmetros e exames). Para conseguirmos obter as restrições mencionadas, o acesso às funções requer login para saber com que utilizador estão a lidar. Se o acesso for autorizado, a função redireciona-o o utilizador para a página *web* relativa aos detalhes pedidos.

De seguida, encontra-se na figura 3.12, a função relativa aos detalhes do médico, a título de exemplo:

```
@user_passes_test(is_medico, login_url='login')
def detalhes_medico(request, medico_id):
    medico = get_object_or_404(Medico, pk=medico_id)
    return render(request, 'detalhes_medico.html', {'medico': medico})
```

Figura 3.12: Função associada aos detalhes do médico

#### 3.4.4 *Home, home\_utente, home\_medico e home\_familiar*

Estas funções permitem ao site acrescentar uma camada de personalização. A função *home* permite com que haja um redirecionamento para a *home\_page* da aplicação enquanto que as outras definem a *home\_page* de cada tipo de utilizador e é onde são mostradas as operações que podem fazer. Estas funções encontram-se definidas na figura 3.13 .



```

def home(request):
    return render(request, "home.html")

@user_passes_test(is_medico, login_url='login')
def home_medico(request):
    nome_utilizador = request.user.nome
    medico_id = request.user.medico.id
    return render(request, 'home_medico.html', {'nome_utilizador': nome_utilizador, 'medico_id': medico_id})

@user_passes_test(is_utente, login_url='login')
def home_utente(request):
    nome_utilizador = request.user.nome
    utente_id = request.user.utente.id
    return render(request, 'home_utente.html', {'nome_utilizador': nome_utilizador, 'utente_id': utente_id})

@user_passes_test(is_familiar, login_url='login')
def home_familiar(request):
    familiar = Familiar.objects.get(email=request.user)
    utente_associado = familiar.id_associado
    nome_utente = utente_associado.nome
    return render(request, 'home_familiar.html', {'utente_associado': utente_associado, "familiar": familiar, "nome_utente": nome_utente})

```

Figura 3.13: Função associada às diferentes home\_page

### 3.4.5 Registar e Login

Estas últimas duas funções têm o intuito de registar utilizadores, e guardar as suas informações na base de dados, e por fim fazer a autenticação dos mesmos. Ambas as funções encontram-se na figura 3.14, apresentada abaixo:

```

def registar_usuario(request):
    if request.user.is_authenticated:
        logout(request)

    tipo_utilizador = request.POST.get("tipo") or request.GET.get("tipo")
    form_mapping = {"U": UtenteCreationForms, "M": MedicoCreationForms, "F": FamiliarCreationForms}

    form_class = form_mapping.get(tipo_utilizador)
    form = form_class(request.POST) if form_class else None

    if request.method == "POST" and form and form.is_valid():
        user = form.save(commit=False)
        user.username = user.email
        user.set_password(form.cleaned_data["password1"])
        user.save()

        return redirect("login")

    context = {"form": form}
    return render(request, "registar.html", context)

def home(request):
    return render(request, "home.html")

@user_passes_test(is_medico, login_url='login')
def home_medico(request):
    nome_utilizador = request.user.nome
    medico_id = request.user.medico.id
    return render(request, 'home_medico.html', {'nome_utilizador': nome_utilizador, 'medico_id': medico_id})

```

Figura 3.14: Funções associadas ao registo dos utilizadores e login de um médico, respetivamente

## 3.5 *Templates*

Na construção de aplicações *web* dinâmicas, a criação de páginas *web* envolve também a apresentação visual ao utilizador. Os *templates*, que são documentos *HTML* especializados, designados para adicionar texto e elementos estruturais às páginas do *website*, desempenham um papel crucial nesse aspecto, permitindo a configuração eficiente do *design* das várias páginas do *site*. Os tópicos seguintes têm o intuito de demonstrar os vários tipos de templates criados, bem como, os resultados da sua implementação.

### 3.5.1 *Template home.html*

A disposição da primeira página do *site* (*home\_page*) que aparece ao utilizador foi configurada no template *home.html* e a sua estética no *home.css*. A disposição da *home\_page* deste site está definida na figura 3.15. Nesta página o utilizador tem a liberdade de escolher se pretende fazer *login* ou registar-se, dependendo do botão onde carrega, sendo assim redirecionado para as respetivas páginas.

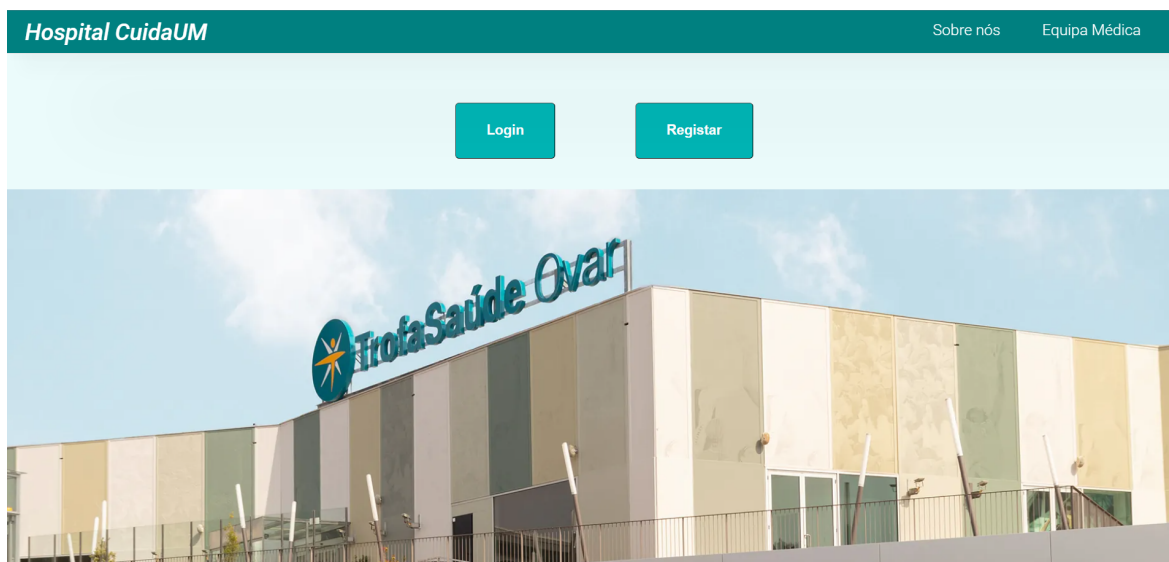


Figura 3.15: Visualização da Home Page

### 3.5.2 *Template registrar.html*

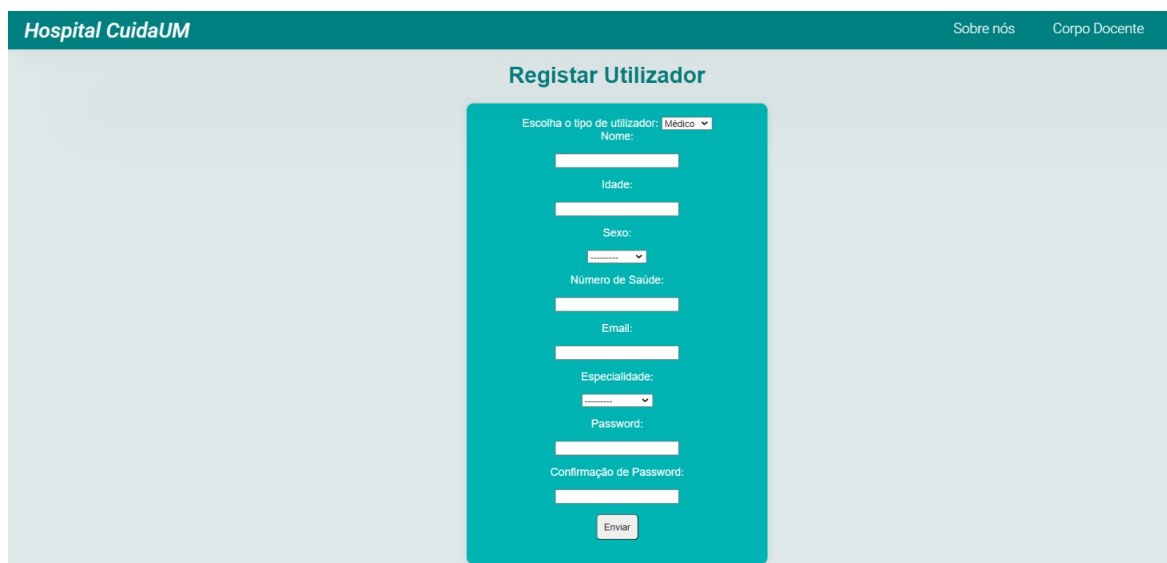
Ao carregar no botão de registar, o utilizador é redirecionado para uma nova página, onde irá seleccionar o seu tipo (utente, médico ou familiar), como apresentado na figura 3.16 abaixo:



The image shows a web form titled "Registrar Utilizador" in a teal box. Inside the box, there is a label "Escolha o tipo de utilizador:" followed by a dropdown menu. The dropdown menu is open, showing four options: "Médico", "Utente", "Médico", and "Familiar". The first "Médico" option is highlighted. To the left of the dropdown is a button labeled "Enviar".

Figura 3.16: Visualização da página de Registo

De seguida, ao carregar no botão Enviar, o utilizador será redirecionado para a página específica relativa ao seu tipo, visto que cada um tem diferentes parâmetros a serem preenchidos no momento de registo. Como exemplo, abaixo encontra-se a figura 3.17, correspondente ao registo de um médico.



The image shows a web form titled "Registrar Utilizador" in a teal box, set against a background with a header "Hospital CuidaUM" and links "Sobre nós" and "Corpo Docente". The form contains the following fields: "Escolha o tipo de utilizador:" with a dropdown menu set to "Médico", "Nome:" with a text input, "Idade:" with a text input, "Sexo:" with a dropdown menu, "Número de Saúde:" with a text input, "Email:" with a text input, "Especialidade:" with a dropdown menu, "Password:" with a text input, and "Confirmação de Password:" with a text input. At the bottom of the form is a button labeled "Enviar".

Figura 3.17: Visualização da página de Registo de um Médico

### 3.5.3 *Template login.html*

Após a realização do registo, o utilizador é automaticamente direcionado para a página de *login*, onde poderá preencher os seus dados de acesso para aceder às restantes funcionalidades. Salienta-se que a autenticação é feita através do *e-mail* e da palavra-passe. Na figura 3.18, encontra-se, a título de exemplo, o *login* do médico registado na subsecção anterior.

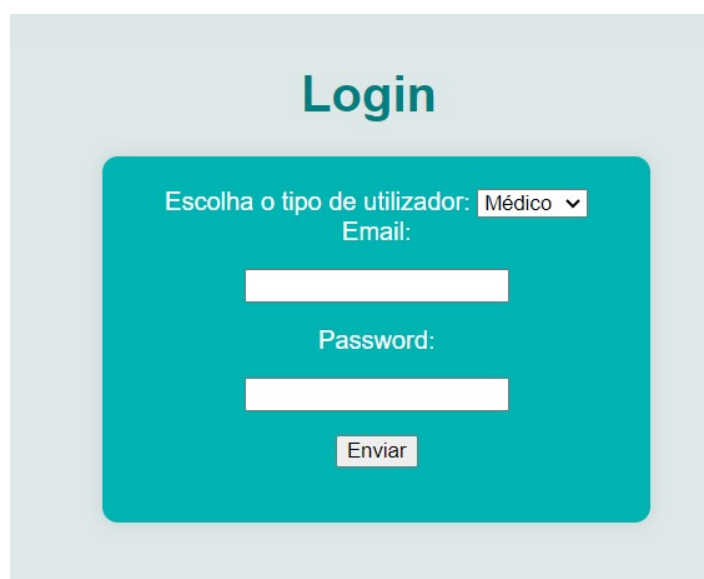


Figura 3.18: Visualização da página de *Login* de um Médico

### 3.5.4 *Templates home\_utente.html, home\_medico.html e home\_familiar.html*

Posteriormente ao *login*, o utilizador acede à *home\_page* específica do seu tipo, tendo estas particularidades e funcionalidades distintas. No que se refere ao utentes e familiares, estes têm um *template* semelhante, no qual podem aceder às informações sobre as consultas, exames, prescrições e parâmetros. Salienta-se uma particularidade na página do familiar, onde existe a indicação do utente a qual este está associado.

Relativamente à *home\_page* do médico, esta é mais complexa que as anteriores devido às permissões que o mesmo tem, isto é, a criação de novas consultas, exames, prescrições e parâmetros dos seus utentes. Além disso, poderá também consultar todas as informações anteriormente mencionadas dos seus utentes e os próprios, a lista de medicamentos presentes na base de dados hospitalar, o corpo clínico do hospital e os familiares associados aos seus pacientes. Na figura 3.19 abaixo mostrada, é possível verificar a constituição da página abordada.

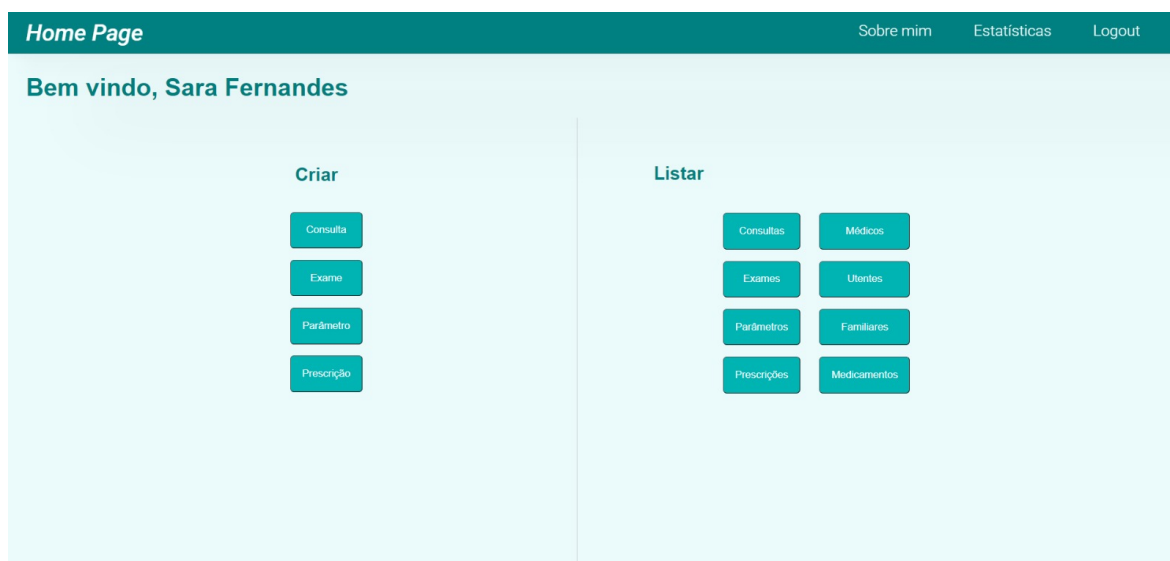
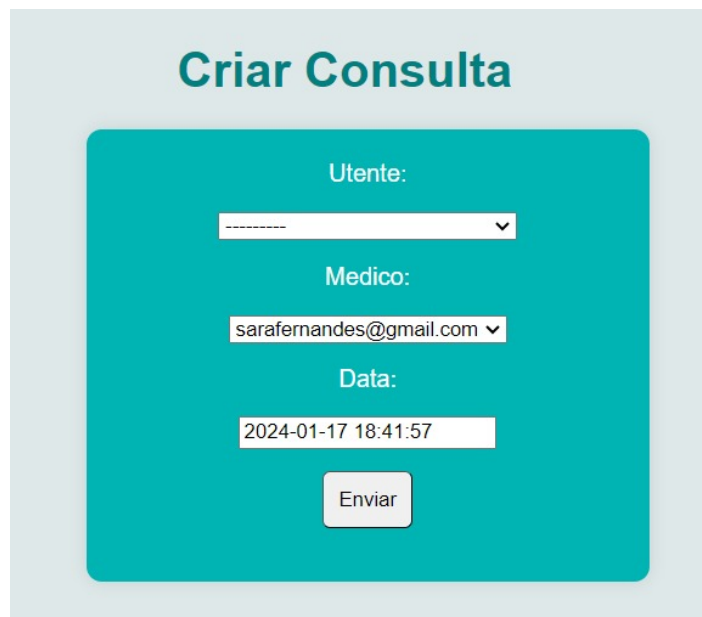


Figura 3.19: Visualização da *home\_page* um Médico

### 3.5.5 *Template* relativos às funções Criar

Relativamente a esta funcionalidade, apenas é permitida aos utilizadores *Médico*, tal como referido anteriormente. Após selecionar o tipo de criação que pretende, o utilizador é redirecionado para uma nova página onde deverá selecionar e preencher diversos parâmetros. Figura 3.20, a página referente à criação de uma nova consulta.



**Criar Consulta**

Utente:  
----- ▾

Medico:  
sarafernandes@gmail.com ▾

Data:  
2024-01-17 18:41:57

Enviar

Figura 3.20: Visualização da página de criação de uma consulta

É importante mencionar, aproveitando o exemplo apresentado, que aquando do preenchimento dos parâmetros necessários para a criação de uma consulta, no campo Médico apenas aparece o e-mail do Médico que está a proceder à criação da consulta, pois cada médico tem de proceder à criação das consultas dos seus próprios utentes, e no campo Data, esta já aparece automaticamente preenchida com a data e hora atual, de forma a facilitar o seu preenchimento com o formato correto. Outra particularidade encontra-se na criação de exames, prescrições e parâmetros, onde cada médico apenas os poderá criar para pacientes aos quais este já deu alguma consulta.

### 3.5.6 *Template* relativos às funções Listar

Estando na *home\_page* de cada um dos tipos de utilizador é possível aceder às listagens permitidas para cada um, referidas no subcapítulo 3.4.1. Ao carregar nos botões relativos, o utilizador é redirecionado para uma página onde se encontram listados todos os registos associados a cada entidade, isto é, utentes, médicos, familiares, consultas, parâmetros, exames, prescrições e medicamentos. Como exemplo, na figura 3.21 encontra-se a página a que o médico é redirecionado ao clicar no botão Consultas do lado esquerdo da *home\_page* apresentada na figura 3.19.

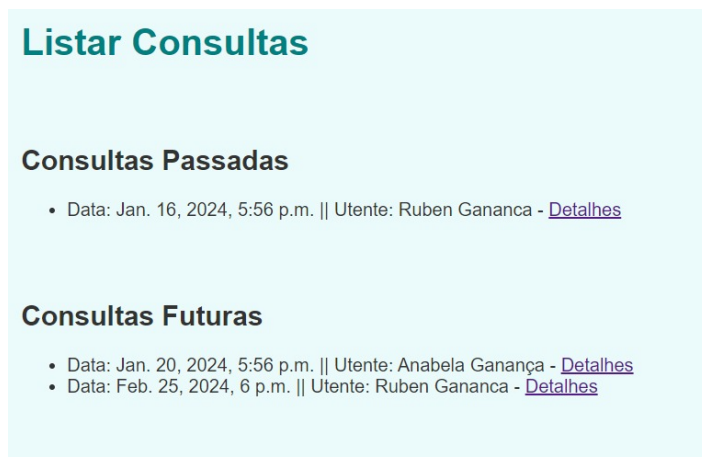


Figura 3.21: Visualização da página de listagem das consultas

### 3.5.7 *Template* relativos às funções Detalhes

Após a listagem de todos os registos associadas à entidade escolhida, é possível aceder aos detalhes de cada registo. Para isso, o utilizador carrega no *link* detalhes, possível de se observar na Figura 3.21, sendo posteriormente redirecionado para a página com os detalhes. Por exemplo, o médico, poderá consultar diversos dados da consulta, tais como: data e hora da consulta, exames pedidos/realizados na consulta, prescrições passadas e parâmetros do paciente medidos nessa mesma consulta. Isto é possível de verificar na Figura 3.22.

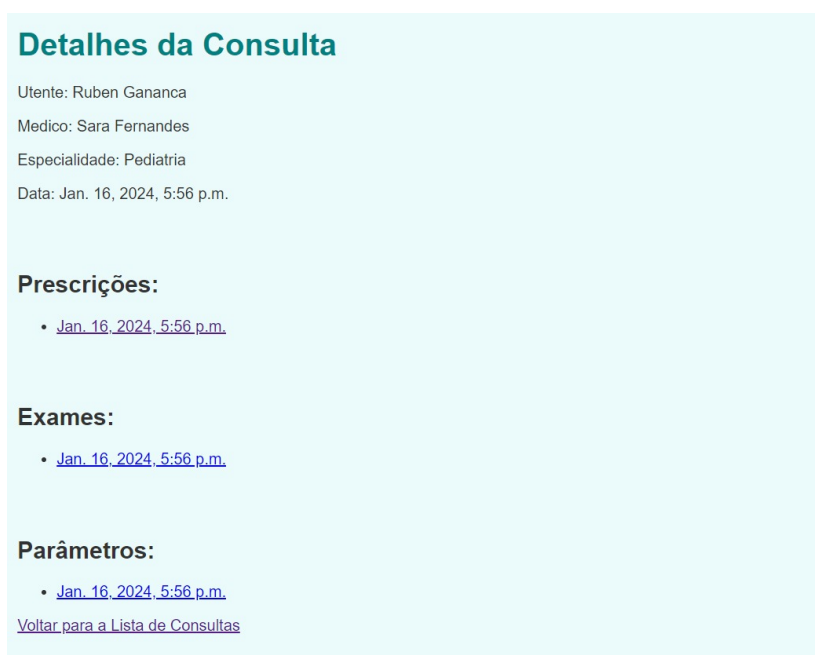


Figura 3.22: Visualização da página de detalhes de uma consulta

## 3.6 URLs

Os *urls* associados à aplicação são essenciais para que um destes endereços definidos fique associado a uma *view* que, por sua vez, retornará uma das páginas *HTML* desenvolvidas.

Em relação à estrutura dos *urlpatterns* definidos, pode-se aferir que o primeiro parâmetro é o endereço que aparece no monitor do utilizador, o segundo é a *view* que a ele está associada e que retornará uma página *HTML* e o terceiro parâmetro é o nome que se quer dar ao endereço de forma a facilitar a sua menção no código-fonte, em vez de menção de todo o endereço. Isto pode ser verificado na figura 3.23.

```
urlpatterns = [  
    path("", home, name="home"),  
    path("home_medico", home_medico, name="home_medico"),  
    path("home_utente", home_utente, name="home_utente"),  
    path("home_familiar", home_familiar, name="home_familiar"),  
  
    path('utentes/', listar_utentes, name='listar_utentes'),  
    path('utentes/<int:utente_id>/', detalhes_utente, name='detalhes_utente'),  
    path('medicos/', listar_medicos, name='listar_medicos'),  
    path('medicos/<int:medico_id>/', detalhes_medico, name='detalhes_medico'),  
    path('familiares/', listar_familiares, name='listar_familiares'),  
    path('familiares/<int:familiar_id>/', detalhes_familiar, name='detalhes_familiar'),  
    path('consultas/', listar_consultas, name='listar_consultas'),  
    path('consultas/<int:consulta_id>/', detalhes_consulta, name='detalhes_consulta'),  
    path('medicamentos/', listar_medicamentos, name='listar_medicamentos'),  
]
```

Figura 3.23: Excerto do documento relativo aos *urls*

## 3.7 Site Admin

Das diversas funcionalidades que o *Django* oferece, uma delas consiste na possibilidade de criar um *site* de administrador de forma automática permitindo diversas funcionalidades, nomeadamente, a verificação dos utilizadores criados, consulta da informação dos mesmos e a capacidade de criar grupos dos utilizadores. Esta última funcionalidade é extremamente pertinente para a realização do projeto, dado que, no momento de criação de um novo *Utilizador*, este é adicionado ao grupo correspondente e, através da *tag* associada ao mesmo, é definido as permissões e acesso restrito a zonas dedicadas. Ou seja, na criação de um



*Médico* e, após este ter realizado o *login* na sua conta, apenas terá acesso às funcionalidades reservadas aos mesmos.

Além disso, no *site Admin* é possível verificar todas as instâncias criadas de cada um dos modelos, bem como manipular as informações de cada um, inclusive, uma eventual eliminação.

## 3.8 Fatores Valorização

Além dos requisitos mínimos apresentados anteriormente, foram ainda solicitados fatores de valorização, pelo que se apresenta de seguida estes fatores.

### 3.8.1 Estatística

De forma a que os médicos tenham a oportunidade de verificar as estatísticas do sistema hospital onde se encontram, eles podem aceder a estas estatísticas através da sua página principal. Esta estatística do hospital apenas está disponível para o médico, uma vez que tem informações gerais do hospital.

Para que esta página e estas informações estivessem disponíveis foi preciso criar uma *view* com este propósito. Deste modo, na função da figura 3.24 é possível verificar que, para o primeiro caso por exemplo, para listar os médicos que deram mais consultas foi necessário adicionar uma anotação com a contagem do número de consultas associadas a cada médico (usando a função *annotate*). Estas contagens depois são ordenadas por ordem decrescente.

```
def estatisticas():
    # Consulta para obter o médico que deu mais consultas
    medico_mais_consultas = Medico.objects.annotate(num_consultas=Count('consulta')).order_by('-num_consultas')[:5]

    # Consulta para obter a quantidade total de consultas dadas por todos os médicos
    total_consultas = Medico.objects.aggregate(total_consultas=Count('consulta'))

    medicamentos_mais_prescritos = Medicamento.objects.annotate(num_prescricoes=Count('prescricao')).order_by('-num_prescricoes')[:5]

    exames_mais_prescritos = Exame.objects.values('tipo_exames').annotate(num_exames=Count('tipo_exames')).order_by('-num_exames')[:5]

    return {
        'medico_mais_consultas': medico_mais_consultas,
        'total_consultas': total_consultas['total_consultas'],
        'medicamentos_mais_prescritos': medicamentos_mais_prescritos,
        'exames_mais_prescritos': exames_mais_prescritos,
    }

@user_passes_test(is_medico, login_url='login')
def estatisticas_view(request):
    medico_id = request.user.medico.id
    estatisticas_data = estatisticas()
    return render(request, 'estatisticas.html', {'estatisticas': estatisticas_data, 'medico_id': medico_id,})
```

Figura 3.24: Função que gera a estatística

A página onde se encontra mostrada esta estatística está na figura 3.25 .

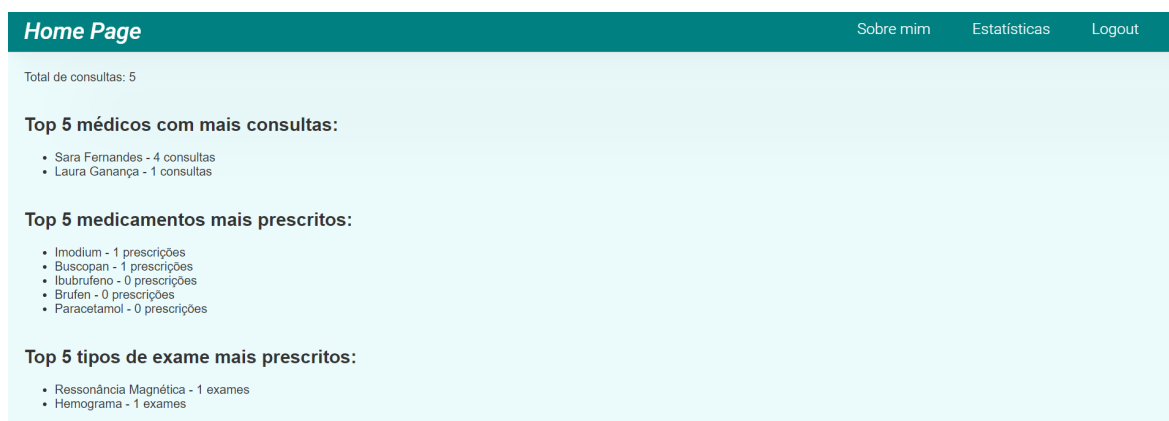


Figura 3.25: Página para visualizar a estatística

### 3.8.2 Registo e Autenticação de utilizadores

A nível de registos e autenticação de utilizadores foi possível criar páginas próprias para os mesmos, em que os utilizadores conseguem se registar e autenticar sem precisar de recorrer ao *admin*. Para além de um registo e login autónomo, se um utilizador tentar entrar numa página que à qual não tem autorização para aceder, vai ser redirecionado para a página de *login*, onde vai verificar a autenticidade do mesmo. Um exemplo deste caso é quando um utente tenta aceder à página de registo de uma consulta, é lhe pedido que volte a fazer login para confirmar se tem efetivamente acesso à página.

## 4. Conclusão

Na realização deste projeto foi possível colocar em prática conhecimentos adquiridos, através da utilização da *framework Django*, bem como aprofundá-los e aprimorá-los. Revelou-se, de facto, uma *framework* com potencialidades para facilitar e auxiliar a construção e implementação de servidores aplicativos.

De uma forma geral, considera-se que os objetivos do trabalho foram atingidos. No entanto, realça-se a dificuldade da atualização do forms no momento em que o médico seleciona o paciente para o qual quer criar exame, prescrição ou parâmetros.

Por fim, e com uma visão futura deste trabalho, pretende-se a implementação de funcionalidades tais como: notificações de agendamento, resultados, entre outras; povoação da base de dados com dados reais; método de pesquisa personalizado.

## ***Referências***

- [1] Django, <https://www.djangoproject.com/>
- [2] Full Stack Python, "Object-relational Mappers (ORMs)", <https://www.fullstackpython.com/object-relational-mappers-orms.html>
- [3] Django, "Documentation, Models" <https://docs.djangoproject.com/en/4.0/topics/db/models/>
- [4] Django, "Documentation, Writing views" <https://docs.djangoproject.com/en/4.0/topics/http/views/>