



Universidade do Minho
Escola de Engenharia

Aplicações Distribuídas

TP2

Aplicação DJANGO



João Miguel da Silva Alves (83624)

Paulo Jorge Alves (84480)

**MESTRADO INTEGRADO EM ENGENHARIA BIOMÉDICA
INFORMÁTICA MÉDICA 2020/2021**

RESUMO

O projeto do segundo trabalho prático é semelhante ao primeiro, contudo com algumas diferenças que foram importantes no momento da implementação. Usou-se a framework DJANGO para realizar este projeto e, conseqüentemente, recorreu-se à linguagem Python.

Esta framework permitiu a elaboração de um sistema que usa uma arquitetura cliente-servidor. O servidor é capaz de responder a pedidos POST e GET por parte de um cliente.

A base de dados usada foi *SQLITE3* o que permitiu guardar todos os dados necessários para poder dar início à aplicação. No lado lógico do servidor, a *Structured Query Language* (SQL) foi usada para comunicar com a base de dados e extrair a data relevante que foi requerida por parte do cliente.

No fim, o objetivo fundamental do sistema é ser capaz de, para além de responder a pedidos *GET* e *POST* também ser capaz de agendar, cancelar e realizar consultas bem como adicionar determinados parâmetros a essas consultas.

Índice

1. Introdução	4
1.1. Contextualização	4
1.2. Apresentação do Caso de Estudo	6
1.3. Objetivos	6
1.4. Estrutura do Relatório	6
2. Levantamento de Requisitos	7
3. Implementação do Projeto	8
3.1. Models	8
3.1.1. Pessoa	8
3.1.2. Utente	8
3.1.3. Médico	9
3.1.4. Funcionário	9
3.1.5. Gestor	10
3.1.6. Medicamento	10
3.1.7. Medição	10
3.1.8. Ficha Médica	11
3.1.9. Consulta	11
3.1.10. Prescrição	12
3.1.11. Exame	12
3.1.12. Entrada Agenda	13
3.2. Forms	14
3.3. Views	15
3.3.1. Login/Logout	15
3.3.2. Utente	17
3.3.3. Funcionário	21
3.3.4. Gestor	24
3.3.5. Médico	27
3.4. Povoamento	35
3.5. Templates	35
3.6. URL's	36
3.7. ADMIN	39
4. Conclusão	40
5. Bibliografia	41
6. Anexos	42

1. Introdução

1.1. Contextualização

A motivação por detrás da conceção de uma framework web está comumente focada na resolução de um ou vários problemas, que poderão abranger diversas áreas ou camadas de uma arquitetura de uma aplicação web, quer na construção de código de servidor, bem como na conceção da componente que será apresentada a um utilizador através de um cliente.

Uma framework para aplicações web é uma framework de software designado para suportar o desenvolvimento de sites web dinâmicos, aplicações Web e serviços Web. Destina-se a aliviar a sobrecarga associada ao desenvolvimento de aplicações Web suportadas por bases de dados^[3].

Assim, surge o Django, uma framework web Python de alto nível que permite o rápido desenvolvimento de sites seguros e de fácil manutenção.

O Django segue o padrão MVT (Model-View-Template). Esta característica promove a reutilização de código e permite que várias interfaces sejam aplicadas sendo assim tudo desenvolvido num só lugar. Caracteriza-se pelas suas filosofias de design, em particular por querer tornar cada elemento da ferramenta independente dos outros, por usar menos, tornando seu desenvolvimento mais rápido e pelo seu design extremamente limpo em todo codificação do seu código, facilitando o acompanhamento da web desenvolvimento^{[2][3]}.

Usou-se o SQLite3 como base de dados para armazenamento dos dados do sistema implementado.

O SQLite é uma biblioteca em linguagem C que implementa um mecanismo de base de dados SQL pequeno, rápido, independente, de alta confiabilidade e com recursos completos. O código-fonte desta base de dados é de domínio público.

Duas a três vezes mais rápido que o MySQL em consultas normais, o SQLite pode chegar a ser até 60 vezes mais rápido que qualquer gerenciador de base de dados. As maiores vantagens do uso desta base de dados, além da elevada velocidade, são o custo e complexidade da aplicação reduzidos, a portabilidade e a acessibilidade, pois está disponível gratuitamente^[1].

Para que fosse possível implementar o sistema proposto foi imprescindível que primeiro se conhecesse a arquitetura do Django (figura 1). Este é construído como um sistema de solicitação-resposta, no qual o navegador faz uma solicitação e o Django invoca uma visualização. Por outro lado, a visualização retorna uma resposta que é enviada ao navegador^[1].

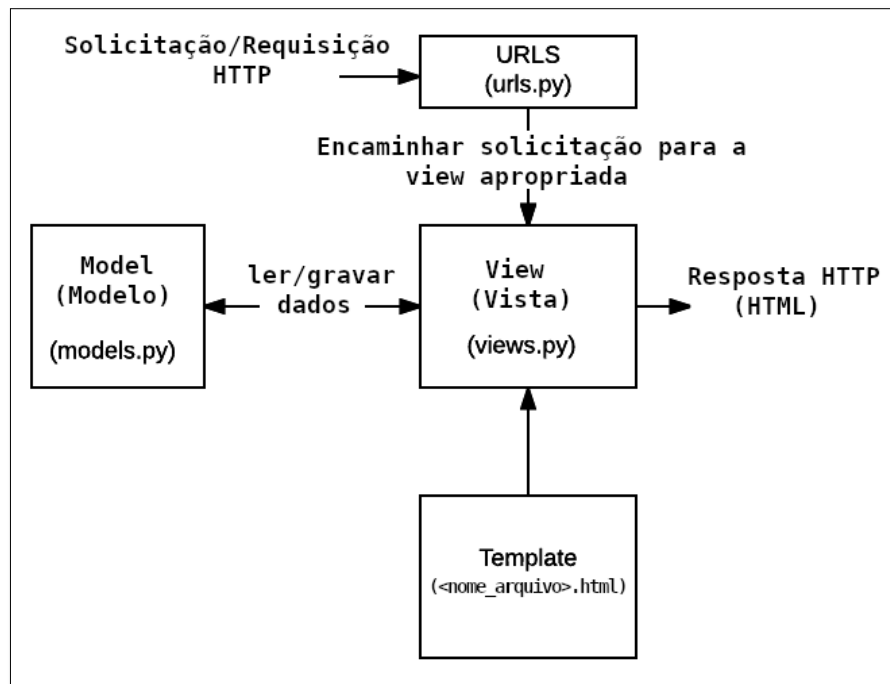


Figura 1 - Arquitetura do Django

O ficheiro *models.py* define a estrutura da base de dados como uma classe Python. Esta classe está disponível para criar, alterar e excluir registos na base de dados. O ficheiro *forms.py* indica os campos de cada uma das classes de modelos que serão tratadas através de formulários de preenchimento pelos utilizadores. O ficheiro *views.py* define a lógica de negócio da página a ser construída. Nesse ficheiro, são definidas as funcionalidades inerentes a cada requisito definido. O arquivo *urls.py* especifica a visualização que será chamada para um determinado padrão de URL. O arquivo *admin.py* permite, não só ver todos os dados da base de dados, como fazer alterações, como alterar ou remover esses dados. Por fim, os templates são arquivos que definem a estrutura ou o layout de uma página HTML, com espaços reservados usados para representar o conteúdo real. Uma *view* pode criar dinamicamente uma página HTML usando um *template* HTML, preenchendo-a com dados de um *model*.

1.2. Apresentação do Caso de Estudo

Para este trabalho foi proposto a implementação de um sistema multiutilizador de gestão de uma clínica (registo de dados em ficha clínica, agenda de consultas, ...).

Para desenvolver o projeto usou-se o Django.

1.3. Objetivos

O principal objetivo deste trabalho é a familiarização com o framework de desenvolvimento de aplicações Django.

1.4. Estrutura do Relatório

Este relatório divide-se fundamentalmente em 3 partes;

Inicialmente, é introduzida uma noção geral sobre o tema. Segue-se o levantamento dos requisitos, e a definição das entidades envolvidas. Por último, a explicação detalhada de todo o projeto.

2. Levantamento de Requisitos

Sendo o objetivo deste trabalho a implementação de um sistema multiutilizador de gestão de uma clínica, achou-se pertinente a utilização de dados relativos a utentes, médicos, funcionários e a gestores da instituição de saúde.

O gestor é o responsável pelo registo de novos médicos e funcionários na instituição. Para adicionar novos médicos, este tem de introduzir o nome do médico, morada, NIF, número do CC, data de nascimento, número da cédula e a respetiva especialidade. Para adicionar novos funcionários, o gestor tem de introduzir o nome do funcionário, morada, NIF, número do CC, data de nascimento e o número de funcionário correspondente.

O funcionário pode registar um utente na IS, introduzindo o nome do utente, morada, NIF, número do CC, data de nascimento, número de utente, número de telefone e email. Desta forma é criada automaticamente uma ficha médica deste novo paciente. Uma vez registado e feito o login, o utente pode agendar uma consulta de determinada especialidade a uma certa hora. A consulta será agendada se, existirem médicos da especialidade para a realizar na hora pretendida, caso contrário não consegue agendar. Esta consulta, passível de ser cancelada até à data da sua realização, fica registada na agenda até ao momento da consulta. Neste momento, o médico inicia a consulta, introduzindo as medições efetuadas durante a consulta e as respetivas observações e prescrições. Caso seja necessário a realização de exames médicos, o médico poderá agendar para uma data posterior à consulta.

Por fim, o gestor tem acesso a outras funcionalidades, como por exemplo ver a lista de utentes, funcionários e médicos da IS. O médico, além do que já foi descrito anteriormente, pode por exemplo, ver a ficha médica de um determinado utente ou consultar a sua lista de consultas. O utente pode consultar as consultas que realizou, tal como os respetivos exames. Isto são apenas alguns exemplos de funcionalidades que os utilizadores do sistema podem ter acesso.

As Entidades envolvidas são: Utente, Médico, Funcionário, Gestor, Consulta, Medicamento, Prescrição, Medição, Exame, Ficha Médica e Entrada de Agenda.

3. Implementação do Projeto

3.1. Models

Um modelo é uma fonte única e definitiva de informações sobre os dados. Contém os campos e comportamentos essenciais dos dados que se está a armazenar. Normalmente, cada modelo é mapeado para uma única tabela da base de dados.

As colunas das tabelas precisam de ter restrições e tipos de dados, para que os modelos do Django tenham opções de campo e tipos de campo (*CharField*, *IntegerField*, *AutoField*, ...) para cada atributo do modelo.

Com tudo, o Django dá uma API de acesso à base de dados gerada automaticamente.

3.1.1. Pessoa

O modelo “Pessoa” é constituído pelos dados pessoais de uma pessoa, considerados relevantes para inserção na base de dados. Tem como atributos: nome, morada, nif (número de identificação fiscal), cc (número do CC) e datanasc (data de nascimento). Dentro da classe Pessoa, existe a classe Meta, que faz com que este modelo não seja usado como tabela, mas sim como parte das tabelas do Utente, Médico e Funcionário.

```

1. class Pessoa(models.Model):
2.     nome = models.CharField(max_length=200, null=False)
3.     morada = models.CharField(max_length=500, null=False)
4.     nif = models.CharField(unique=True, max_length=9, null=False)
5.     cc = models.CharField(max_length=12, null=False)
6.     datanasc = models.DateField(null=False)
7.
8.     class Meta:
9.         abstract = True

```

3.1.2. Utente

O modelo "Utente" consiste nas informações pessoais do paciente consideradas relevantes para inserção na base de dados. A chave primária é o id do utente, tem também o número de utente, telefone, telefone de emergência e email. Como referido anteriormente, a classe Utente está ligada à classe Pessoa, ou seja, os atributos da Pessoa continuam a ser atributos do Utente.


```

1. class Utente(Pessoa):
2.     id_utente = models.AutoField(primary_key=True, null=False)
3.     numutente = models.CharField(unique=True, max_length=200, null=False)
4.     telefone = models.CharField(max_length=14, null=False)
5.     telefone_emergencia = models.CharField(max_length=14, null=False)
6.     email = models.CharField(max_length=200, null=False)
7.
8.     def __str__(self):
9.         return self.nome

```

3.1.3. Médico

O modelo "Médico" consiste nas informações pessoais do médico consideradas relevantes para inserção na base de dados. A chave primária é o id do médico, tem também a cédula médica e a respetiva especialidade. Como referido anteriormente, a classe Médico está ligada à classe Pessoa, ou seja, os atributos da Pessoa continuam a ser atributos do Médico.

```

1. class Medico(Pessoa):
2.     id_medico = models.AutoField(primary_key=True, null=False)
3.     cedula = models.CharField(unique=True, max_length=200, null=False)
4.     especialidade = models.CharField(max_length=200, null=False)
5.
6.     def __str__(self):
7.         return str(self.nome) + ":" + str(self.morada) + ":" + str(self.nif) + ":" +
str(self.cc) + ":" + str(self.datanasc) + ":" + str(self.cedula) + ":" +
str(self.especialidade)

```

3.1.4. Funcionário

O modelo "Funcionário" consiste nas informações pessoais do funcionário consideradas relevantes para inserção na base de dados. A chave primária é o id do funcionário e tem também o respetivo número de funcionário. Como referido anteriormente, a classe Funcionário está ligada à classe Pessoa, ou seja, os atributos da Pessoa continuam a ser atributos do Funcionário.

```

1. class Funcionario(Pessoa):
2.     id_funcionario = models.AutoField(primary_key=True, null=False)
3.     numfunc = models.IntegerField(unique=True, null=False)
4.
5.     def __str__(self):
6.         return str(self.nome) + ":" + str(self.morada) + ":" + str(self.nif) + ":" +
str(self.cc) + ":" + str(self.datanasc) + ":" + str(self.numfunc)

```

3.1.5. Gestor

O modelo "Gestor" consiste nas informações pessoais do gestor consideradas relevantes para inserção na base de dados. A chave primária é o id do gestor, tem também o telefone, telefone de emergência e email. Como referido anteriormente, a classe Gestor está ligada à classe Pessoa, ou seja, os atributos da Pessoa continuam a ser atributos do Gestor.

```

1. class Gestor(Pessoa):
2.     id_gestor = models.AutoField(primary_key=True, null=False)
3.     telefone = models.CharField(max_length=14, null=False)
4.     telefone_emergencia = models.CharField(max_length=14, null=False)
5.     email = models.CharField(max_length=200, null=False)
6.
7.     def __str__(self):
8.         return self.nome

```

3.1.6. Medicamento

O modelo "Medicamento" consiste nas informações relacionadas com o medicamento consideradas relevantes para inserção na base de dados. A chave primária é o id do medicamento, tem também o dci, o nome, a forma farmacêutica, a dosagem, o estado de autorização, se é ou não genérico e o titular aim.

```

1. class Medicamento(models.Model):
2.     id_medicamento = models.AutoField(primary_key=True, null=False)
3.     dci = models.CharField(max_length=200, null=False)
4.     nome = models.CharField(max_length=200, null=False)
5.     formafarmacêutica = models.CharField(max_length=200, null=False)
6.     dosagem = models.CharField(max_length=200, null=True)
7.     estadoautorizacao = models.CharField(max_length=200, null=False)
8.     generico = models.BooleanField(default=True)
9.     titular_aim = models.CharField(max_length=200, null=False)
10.
11.     def __str__(self):
12.         return str(self.id_medicamento) + ":" + str(self.dci) + ":" + str(self.nome) +
            ":" + str(self.formafarmacêutica) + ":" + str(self.dosagem) + ":" +
            str(self.estadoautorizacao) + ":" + str(self.generico) + ":" + str(self.titular_aim)

```

3.1.7. Medição

O modelo "Medição" consiste nas informações relacionadas com a medição realizada em cada consulta. A chave primária é o id da medição, tem também a data, o peso, a glicemia, a altura (cm), a tensão arterial, o colesterol, os triglicérideos, a saturação e por fim, o inr.

```

1. class Medicao(models.Model):
2.     id_medicao = models.AutoField(primary_key=True, null=False)
3.     data = models.DateField(null=False)
4.     peso = models.DecimalField(max_digits=200, decimal_places=5, null=False)
5.     glicemia = models.DecimalField(max_digits=200, decimal_places=5, null=False)
6.     altura = models.IntegerField(null=False)
7.     tensaoarterial = models.IntegerField(null=False)
8.     colesterol = models.IntegerField(null=False)
9.     trigliceridios = models.IntegerField(null=False)
10.    saturacao = models.IntegerField(null=False)
11.    inr = models.IntegerField(null=False)
12.
13.    def __str__(self):
14.        return str(self.id_medicao) + ":" + str(self.data) + ":" + str(self.peso) + ":"
+ str(self.glicemia) + ":" + str(self.altura) + ":" + str(self.tensaoarterial) + ":" +
+ str(self.colesterol) + ":" + str(self.trigliceridios) + ":" + str(self.saturacao) + ":"
+ str(self.inr)

```

3.1.8. Ficha Médica

O modelo "Ficha Médica" consiste nas informações relacionadas com a ficha médica de cada utente. A chave primária é o id da ficha médica e tem também o histórico do estado de saúde do mesmo utente.

Este modelo também tem um atributo *OneToOneField* que permite ao sistema estabelecer uma relação de 1:1 entre este modelo e o modelo Utente, ou seja, cada utente só tem uma ficha médica, e vice-versa. Além deste atributo, tem também um atributo *ManyToManyField* que relaciona (N:N) este modelo com o modelo Medição, ou seja, cada ficha médica pode ter mais que uma medição e vice-versa.

```

1. class Ficha_medica(models.Model):
2.     id_ficha = models.AutoField(primary_key=True, null=False)
3.     utente = models.OneToOneField(Utente, on_delete=models.CASCADE)
4.     medicoes = models.ManyToManyField(Medicao)
5.     historico = models.CharField(max_length=2000, blank = True)
6.
7.     def __str__(self):
8.         return str(self.id_ficha) + ":" + str(self.utente) + ":" + str(self.historico)

```

3.1.9. Consulta

O modelo "Consulta" consiste nas informações relacionadas com as consultas. A chave primária é o id da consulta e tem também a data e as observações efetuadas.

Este modelo também tem dois atributos *ForeignKey* que permite ao sistema estabelecer uma relação de 1:N entre este modelo e os modelos Médico e Ficha Médica, ou seja, cada

consulta é efetuada por um único médico e para um único utente (que tem a correspondente ficha médica). No entanto, um médico pode efetuar muitas consultas, tal como o utente pode ir a muitas consultas.

```

1. class Consulta(models.Model):
2.     id_consulta = models.AutoField(primary_key=True, null=False)
3.     data = models.DateField(null=True)
4.     observacoes = models.CharField(max_length=2000, blank = True)
5.     medico = models.ForeignKey(Medico, on_delete=models.CASCADE, null = True)
6.     fichamedica = models.ForeignKey(Ficha_medica, on_delete=models.CASCADE, null=True)
7.
8.     def __str__(self):
9.         return str(self.id_consulta) + ":" + str(self.data) + ":" + str(self.medico) +
           ":" + str(self.observacoes)

```

3.1.10. Prescrição

O modelo "Prescrição" consiste nas informações relacionadas com as prescrições receitadas nas consultas. A chave primária é o id da prescrição e tem também a data e a toma.

Este modelo também tem um atributo *ForeignKey* que permite ao sistema estabelecer uma relação de 1:N entre este modelo e o modelo Consulta, ou seja, cada consulta só pode ter uma prescrição, no entanto, a mesma prescrição pode ser receitada em diversas consultas. Além deste atributo, tem também um atributo *ManyToManyField* que relaciona (N:N) este modelo com o modelo Medicamentos, ou seja, cada prescrição pode ter vários medicamentos e vice-versa.

```

1. class Prescricao(models.Model):
2.     id_prescricao = models.AutoField(primary_key=True, null=False)
3.     data = models.DateField(null=False)
4.     med = models.ManyToManyField(Medicamento)
5.     toma = models.CharField(max_length=200, null=False)
6.     consulta = models.ForeignKey(Consulta, on_delete=models.CASCADE, null=True)
7.
8.     def __str__(self):
9.         return str(self.id_prescricao) + ":" + str(self.med) + ":" + str(self.data) +
           ":" + str(self.toma) + ":" + str(self.consulta)

```

3.1.11. Exame

O modelo "Exame" consiste nas informações relacionadas com os exames marcados nas consultas. A chave primária é o id da exame e tem também o tipo, o local, a data e hora, a duração, o preço, o estado e as observações necessárias para a realização do mesmo.

Este modelo também tem um atributo *ForeignKey* que permite ao sistema estabelecer uma relação de 1:N entre este modelo e o modelo Consulta, ou seja, cada exame corresponde a apenas uma consulta. No entanto, numa consulta podem ser marcados diversos exames.

```

1. class Exame(models.Model):
2.     id_exame = models.AutoField(primary_key=True, null=False)
3.     tipo = models.CharField(max_length=200, null=False)
4.     local = models.CharField(max_length=200, null=False)
5.     data_hora = models.DateTimeField(null=False)
6.     duracao = models.TimeField()
7.     preco = models.DecimalField(max_digits=200, decimal_places=5, null=False)
8.     estado = models.BooleanField(max_length=200, null=False)
9.     observacoes = models.CharField(max_length=200, null=False)
10.    consulta = models.ForeignKey(Consulta, on_delete=models.CASCADE, null=True)
11.
12.    def __str__(self):
13.        return str(self.id_exame) + ":" + str(self.tipo) + ":" + str(self.data_hora) +
        ":" + str(self.observacoes)

```

3.1.12. Entrada Agenda

A modelo "Entrada Agenda" consiste nas informações relacionadas com as consultas marcadas. A chave primária é o id da entrada da agenda e tem também a data e hora da consulta e o estado da mesma, que ao ser marcada fica como "Marcado".

Este modelo também tem dois atributos *ForeignKey* que permite ao sistema estabelecer uma relação de 1:N entre este modelo e os modelos Médico e Utente, ou seja, cada consulta é marcada com um único médico e para um único utente. No entanto, um médico pode estar em muitas consultas, tal como o utente pode marcar mais que uma consulta.

```

1. class Entrada_Agenda(models.Model):
2.     id_entradagenda = models.AutoField(primary_key=True, null=False)
3.     data_hora = models.DateTimeField(null=False)
4.     estado = models.CharField(max_length=20)
5.     utente = models.ForeignKey(Utente, on_delete=models.CASCADE)
6.     medico = models.ForeignKey(Medico, on_delete=models.CASCADE)
7.
8.     def __str__(self):
9.         return str(self.utente) + ":" + str(self.medico)

```

3.2. Forms

Uma *Django Form* é responsável por aceitar um determinado input por parte do utilizador, validado e passá-lo para objetos *python*.

As forms “*NewUserForm*” e “*UserForm*” permitem ao utilizador autenticar as suas credenciais ou se não possuir uma conta, então pode registar-se com as suas novas credenciais.

```

1. class NewUserForm(UserCreationForm):
2.     username = forms.CharField(max_length=50)
3.     password1 = forms.CharField(max_length=20)
4.     password2 = forms.CharField(max_length=20)
5.     email = forms.EmailField(required=True)
6.
7.     class Meta:
8.         model = User
9.         db_table = 'auth_user'
10.        fields = ("username", "email", "password1", "password2")
11.
12.    def save(self, commit=True):
13.        user = super(NewUserForm, self).save(commit=False)
14.        user.email = self.cleaned_data['email']
15.        if commit:
16.            user.save()
17.        return user
18.
19. class UserForm(AuthenticationForm):
20.     #email = forms.EmailField(required=True)
21.
22.     class Meta:
23.         model = User
24.         db_table = 'auth_user'
25.         fields = ("username", "password")
26.    def save(self, commit=True, *args, **kwargs):
27.        user = super(UserForm, self).save(commit=False, *args, **kwargs)
28.        #user.email = self.cleaned_data['email']
29.        if commit:
30.            user.save()
31.        return user

```

A form “*MarcarConsultaForm*” é necessária para o utente poder submeter o seu número de utente, a especialidade que pretende ter consulta e a data e hora que são necessárias para marcar a consulta.

```

1. class MarcarConsultaForm(forms.Form):
2.     especialidade = forms.CharField(label = "Especialidade")
3.     data = forms.DateField(label="Data")
4.     hora = forms.TimeField(label = "Hora")

```

As restantes forms são análogas a esta, apenas diferem no que vai ser necessário ser submetido que irá depender da ação do utilizador em causa. Por essa razão encontram-se em anexo.

3.3. Views

Uma função *view* é uma função Python que recebe um pedido WEB e retorna uma resposta WEB. Esta resposta poderá ser uma página com conteúdo HTML, um *redirect* para outra página ou mesmo um *erro 404*.

A *view* contém o processamento lógico que será necessário para retornar essa resposta. A função irá pedir a informação proveniente de um modelo previamente criado e passá-la para um *template*.

3.3.1. Login/Logout

Para se conseguir realizar *login*, *logout* ou até um registo de conta, tivemos de criar as seguintes *views*.

Os *logins* e registos para o utente, medico, gestor e funcionário são os mesmos com a exceção de que ao conseguirem dar *login*, cada um irá para a página respetiva. Os excertos não apresentados abaixo encontram-se em anexo.

Para conseguirem entrar é necessário terem uma conta. Caso contrário, terão de se registar, inserindo um *username*, email e uma password. Este username terá que corresponder a uma letra (U,F,G ou M) com o respetivo número pessoal. Caso não estejam registados na base de dados, não se poderão registar na app. Este será um pedido POST caso o utilizador coloque as suas credenciais válidas e irá retornar uma página com as ações que cada um pode realizar. Se não for um pedido POST, então irá aparecer uma página para o utilizador colocar as suas credenciais.

```

1. from django.shortcuts import render
2.
3. # Create your views here.
4.
5. from django.contrib import auth
6. from django.contrib.auth import authenticate, login, logout
7. from django.contrib.auth.decorators import login_required, user_passes_test
8. from django.contrib.auth.forms import UserCreationForm, AuthenticationForm
9. from django.core.checks import messages
10. from django.contrib import messages
11. from django.shortcuts import render, redirect
12. from django.http import HttpResponse, HttpResponseRedirect
13. from django.utils.regex_helper import Group
14. from django.contrib.auth.forms import UserCreationForm
15. from django.contrib.auth import login, logout, authenticate
16. from .forms import *
17. from datetime import *
18. import datetime

```

```

19. from .models import *
20.
21. # pagina inicial
22. from django.urls import reverse
23.
24. # from clinica.forms import SignUpForm
25. from .models import *
26.
27. def homepage_view(request):
28.     return render(request, "inicial.html")
29.
30. def login_utente(request):
31.     if request.method == "POST":
32.         form = AuthenticationForm(request, data=request.POST)
33.
34.         if form.is_valid():
35.             username = form.cleaned_data.get('username')
36.             password = form.cleaned_data.get('password')
37.             user = authenticate(username=username, password=password)
38.             if user is not None:
39.                 login(request, user)
40.                 messages.info(request, f"you are now logged in as {username} ")
41.                 return redirect('/alves_lda/utente/'+username)
42.             else:
43.                 messages.error(request, "Invalid username or password")
44.         else:
45.             messages.error(request, "Invalid username or password")
46.         form = AuthenticationForm()
47.         return render(request, 'utente/login_utente.html', {'form': form})
48.
49.
50. def registrar_utente(request):
51.     if request.method == "POST":
52.         form = NewUserForm(request.POST)
53.
54.         if form.is_valid():
55.             user = form.save()
56.             username = form.cleaned_data.get('username')
57.             x=username.replace("U","")
58.             if "U" in username and Utente.objects.filter(numutente=x).exists():
59.                 messages.success(request, f"New Account Created: {username}")
60.                 login(request, user)
61.                 messages.info(request, f"You are now logged in as {username}")
62.                 return redirect("/alves_lda/")
63.             else:
64.                 for msg in form.error_messages:
65.                     messages.error(request, f"{msg}: {form.error_messages[msg]}")
66.             form = NewUserForm()
67.             return render(request, "utente/registrar_utente.html", context={"form": form})

```

Qualquer utilizador pode efetuar logout, no qual irá sair da sua conta e passará para a página inicial da aplicação.

```

1. def sair(request):
2.     logout(request)
3.     messages.info(request, "Logged out successfully")
4.     return redirect("/alves_lda")

```


3.3.2. Utente

Quando o utilizador conseguir realizar o seu login, a *view* seguinte será chamada e irá aparecer no browser uma página com todas as ações que o utente pode realizar. Este pode marcar uma consulta, onde tem acesso a todas as especialidades existentes na clínica, ver todas as suas consultas marcadas, ver todas as consultas e exames já feitos, cancelar uma consulta e fazer logout.

```
1. @login_required
2. def utente(request,username):
3.     return render(request, 'utente/utente.html', {'username': username})
```

Quando o utente carrega em marcar consulta, o utente tem acesso a todas as especialidades existentes na clínica, assim como o horário de funcionamento da mesma.

```
1. @login_required
2. def horario(request, username):
3.     return render(request, 'utente/horario.html', {'username':username})
4.
5. @login_required
6. def especialidades(request, username):
7.     lista=[]
8.     medicos = Medico.objects.all()
9.     for medico in medicos:
10.         a=medico.especialidade
11.         if a not in lista:
12.             lista.append(a)
13.     return render(request, 'utente/especialidades.html', {'lista':lista,'username':
        username})
```

Para marcar a consulta, o utente terá de submeter a especialidade que pretende ter uma consulta tal como a data e a hora. Ao fazê-lo, a *view* “marcar” irá ser chamada e ocorrerá este processamento. Em primeiro lugar, verifica se a data pretendida não é anterior à do momento da marcação. Se não for, irá verificar se os minutos que o utente submeteu são diferentes de 0 ou 30, uma vez que só se pode realizar uma consulta de meia em meia hora, entre as 09H, inclusive e as 18H, exclusive e não pode marcar entre as 13H, inclusive e as 14H, exclusive. Além disso, não pode marcar consultas aos Domingos. Caso os dados introduzidos sejam válidos, então vão-se buscar todos os médicos com a especialidade submetida. De seguida, irá buscar todas as consultas marcadas com esse médico. Se o médico não tiver consultas marcadas, então vai-se criar um objeto de entrada de agenda que corresponde a uma consulta marcada. Caso o médico tenha consultas marcadas, então irá verificar para cada uma delas se coincide com a que foi submetida. Para tal verifica se a data e a hora são as mesmas e, assim, o médico está ocupado e passa para outro médico da mesma especialidade. Caso não

haja uma consulta marcada para essa data e hora no respectivo médico, então irá ser criado um objeto na entrada de agenda, se não aparece uma página a afirmar que não é possível marcar uma consulta com essas credenciais.

```

1. @login_required
2. def marcar_consulta(request, username):
3.     form = MarcarConsultaForm()
4.     return render(request, 'utente/marcar_consulta.html', context={'form': form
5.     'username':username })
6. @login_required
7. def marcar(request, username):
8.
9.     x=username.replace("U", "")
10.    utente = Utente.objects.get(numutente=x)
11.    especialidade = request.POST['especialidade']
12.    data = request.POST['data']
13.    hora = request.POST['hora']
14.    data1 = data.split("-")
15.    hora1 = hora.split(":")
16.
17.    dt1 = date(int(data1[0]), int(data1[1]), int(data1[2]))
18.    tm1 = time(int(hora1[0]), int(hora1[1]), int(hora1[2]))
19.
20.    if dt1 >= datetime.now().date():
21.        if tm1.minute != 0 and tm1.minute != 30:
22.            string = "Introduza uma hora válida! A Hora deverá ter o formato ##:00:00H
23.            ou ##:30:00H, sendo ## a hora pretendida."
24.            return render(request, "utente/erros2.html", {'string': string})
25.        else:
26.            if (tm1.hour < 13 or tm1.hour >= 14) and (tm1.hour >= 9 and tm1.hour < 18 )
27.            and (dt1.weekday() != 6):
28.                medicos = Medico.objects.filter(especialidade=especialidade)
29.
30.                for medico in medicos:
31.                    agenda = Entrada_Agenda.objects.filter(medico=medico)
32.                    if agenda == None:
33.                        data1 = data.split("-")
34.                        hora1 = hora.split(":")
35.                        dt = datetime(int(data1[0]), int(data1[1]), int(data1[2]))
36.                        tm = time(int(hora1[0]), int(hora1[1]), int(hora1[2]))
37.                        combined = dt.combine(dt, tm)
38.                        entradaagenda = Entrada_Agenda.objects.create(data_hora=
39.                        combined, estado="Marcado", utente=utente, medico=medico)
40.                        entradaagenda.save()
41.                        string = "Consulta Marcada com: " + medico.nome
42.                        return render(request, "utente/erro.html", context={'string':
43.                        string, 'username':username})
44.                    else:
45.                        contador = 0
46.                        contador1 = 0
47.                        for eag in agenda:
48.                            contador1+=1
49.                            if eag.data_hora.date() == dt1:
50.                                if eag.data_hora.time() == tm1:
51.                                    break
52.                                else:
53.                                    contador+=1
54.                            else:
55.                                contador+=1
56.
57.                        if contador == contador1:
58.                            data1 = data.split("-")

```

```

55.         hora1 = hora.split(":")
56.         dt = datetime(int(data1[0]), int(data1[1]), int(data1[2]))
57.         tm = time(int(hora1[0]), int(hora1[1]), int(hora1[2]))
58.         combined = dt.combine(dt, tm)
59.         entradaagenda = Entrada_Agenda.objects.create(data_hora=
combined, estado="Marcado", utente=utente, medico=medico)
60.         entradaagenda.save()
61.         string = "Consulta Marcada com: " + medico.nome
62.         return render(request, "utente/erro.html", context={'string'
: string, 'username':username})
63.     else:
64.         string = "Não é possível marcar consulta neste horário! Verifique o
horário de atendimento e remarque..."
65.         return render(request, "utente/erros2.html", {'string': string})
66.         string = "Não é possível marcar consulta! Remarque..."
67.         return render(request, "utente/erros2.html", {'string': string})
68.     else:
69.         string = "Não é possível marcar uma consulta no passado! Remarque..."
70.         return render(request, "utente/erros2.html", {'string': string})

```

Uma outra ação que o utente pode realizar é ver todas as suas consultas marcadas. Para tal é chamada a view “veragenda” que retornará as consultas marcadas desse utente.

```

1. @login_required
2. def veragenda(request, username):
3.     x = username.replace("U", "")
4.     utente = Utente.objects.get(numutente=x)
5.
6.     if Entrada_Agenda.objects.filter(utente=utente).exists():
7.         entradas_agenda = Entrada_Agenda.objects.filter(utente=utente)
8.         return render(request, 'utente/ver_agenda.html', context={'consultas':
entradas_agenda, 'username':username})
9.     else:
10.         string = "Não tem consultas marcadas!"
11.         return render(request, "medico/erros2.html", {'string': string})

```

Outra ação que o utente pode realizar é ver todas as suas consultas realizadas. Para tal é chamada a view “ver_hist_consultas”, que retornará as consultas desse utente através da sua ficha médica.

```

1. @login_required
2. def historico_consultas(request, username):
3.     x = username.replace("U", "")
4.     utente = Utente.objects.get(numutente=x)
5.     fichamedica = Ficha_medica.objects.get(utente = utente)
6.
7.     if Consulta.objects.filter(fichamedica = fichamedica).exists():
8.         consultas = Consulta.objects.filter(fichamedica = fichamedica)
9.         return render(request, "utente/ver_hist_consultas.html", {'consultas':consultas,
'username':username})
10.    else:
11.        string = "Utente introduzido não tem consultas realizadas!"
12.        return render(request, "gestor/erros2.html", {'string': string})

```

Outra ação do utente é poder ver informações sobre todos os seus exames. Para tal, usa-se a view “ver_hist_exames”, que irá buscar todas as consultas da respetiva ficha médica. Para cada consulta irá buscar todos os exames que foram requisitados e colocam numa lista. Por fim, essa lista é retornada para o *template* que fornecerá as informações dos exames.

```

1. @login_required
2. def historico_exames(request, username):
3.     x = username.replace("U", "")
4.     lista_exames = []
5.     utente = Utente.objects.get(numutente=x)
6.     fichamedica = Ficha_medica.objects.get(utente = utente)
7.
8.     if Consulta.objects.filter(fichamedica=fichamedica).exists():
9.         consultas = Consulta.objects.filter(fichamedica = fichamedica)
10.
11.         for consulta in consultas:
12.             exames = Exame.objects.filter(consulta = consulta)
13.             if exames:
14.                 lista_exames.append(exames)
15.
16.         if len(lista_exames)==0:
17.             string = "Utente introduzido não tem exames!"
18.             return render(request, "gestor/erros2.html", {'string': string})
19.         else:
20.             return render(request, "utente/ver_hist_exames.html", context={'exames':
21. lista_exames, 'username':username})
22.     else:
23.         string = "Utente introduzido não tem consultas realizadas!"
24.         return render(request, "gestor/erros2.html", {'string': string})

```

O utente pode também cancelar uma consulta e para tal recorre-se à view “cancelar_consulta” que retornará uma página onde o utente tem de colocar a especialidade, bem como a data e hora da consulta que pretende desmarcar. De seguida, através da view “cancelar”, irá ver para cada entrada na tabela “Entrada_Agenda” se existe alguma com as credenciais submetida. Se assim acontecer, então é cancelada e retirada da tabela, caso contrário aparece uma página a dizer que não poderá cancelar a consulta, pois ainda não a marcou.

```

1. @login_required
2. def cancelar_consulta(request, username):
3.     form = MarcarConsultaForm()
4.     return render(request, 'utente/cancelar_consulta.html', context={'form': form,
5. 'username':username})
6.
7. @login_required
8. def cancelar(request, username):
9.     x = username.replace("U", "")
10.    utente = Utente.objects.get(numutente=x)
11.    especialidade = request.POST['especialidade']
12.    data = request.POST['data']
13.    hora = request.POST['hora']

```

```

13. Entradas_Agenda = Entrada_Agenda.objects.all()
14. data1 = data.split("-")
15. hora1 = hora.split(":")
16.
17. dt = date(int(data1[0]), int(data1[1]), int(data1[2]))
18. tm = time(int(hora1[0]), int(hora1[1]), int(hora1[2]))
19.
20. for eag in Entradas_Agenda:
21.     if utente == eag.utente and dt == eag.data_hora.date() and tm ==
eag.data_hora.time() and especialidade == eag.medico.especialidade:
22.         eag.delete()
23.         string = "Consulta cancelada com sucesso!"
24.         return render(request, "utente/erro.html", context={'string' : string,
'username':username})
25.
26. string = "Nao marcou essa consulta para a poder cancelar..."
27. return render(request, "utente/erro.html", context={'string' : string, 'username':
username})

```

3.3.3. Funcionário

O funcionário será outro utilizador da aplicação e pode efetuar as seguintes ações: adicionar um utente à base de dados, ver as consultas marcadas para um determinado utente, ver utentes que utilizam ou utilizaram um determinado medicamento e pode também ver toda a informação de um determinado utente.

```

1. @login_required
2. def funcionario(request,username):
3.     return render(request, 'funcionario/funcionário.html', {'username': username})

```

Para adicionar um utente recorre-se às seguintes *views*. A primeira *view* retorna uma página onde aparecem todas as informações que o funcionário necessita de submeter para conseguir adicionar um utente à base de dados. De seguida, com o recurso à *view* “submeter_u”, irá buscar todas as informações que foram submetidas antes e irá realizar a criação de uma nova entrada na tabela “Utente” com essas informações.

```

1. @login_required
2. def adicionar_u(request, username):
3.     form = UtenteForm()
4.     return render(request, 'funcionario/adicionar_utente.html', context={'form': form,
'username':username})
5.
6. @login_required
7. def submeter_u(request, username):
8.     nome = request.POST["nome"]
9.     morada = request.POST["morada"]
10.    nif = request.POST["nif"]
11.    cc = request.POST["cc"]
12.    datanasc = request.POST["datanasc"]
13.    numutente = request.POST["numutente"]

```

```

14.     telefone = request.POST["telefone"]
15.     telefone_emergencia = request.POST["telefone_emergencia"]
16.     email = request.POST["email"]
17.
18.     utente = Utente.objects.create(nome = nome, morada = morada, nif = nif, cc = cc,
    datanasc = datanasc, numutente = numutente, telefone = telefone, telefone_emergencia=
    telefone_emergencia, email = email)
19.
20.     utente.save()
21.
22.     fichamedica = Ficha_medica(utente=utente)
23.     fichamedica.save()
24.
25.     string = "Utente adicionado!"
26.     return render(request, "funcionario/erro.html", {'string':string, 'username':username})

```

O funcionário pode ainda ver as consultas que estão marcadas para um determinado utente. Para tal, recorre-se à *view* “entrada_agenda” que irá retornar uma página onde será necessário submeter o número do utente. De seguida, com a *view* “ver_entrada_agenda”, o servidor vai buscar o utente com esse número e irá buscar todas as entradas na tabela “Entrada_Agenda” que contêm esse utente. Por fim, retorna uma página onde aparecem as informações sobre essas consultas marcadas como: o id dessa consulta marcada, o médico e a data e hora dessa consulta.

```

1. @login_required
2. def entrada_agenda(request,username):
3.     form = HistoricoConsultasForm()
4.     return render(request, 'funcionario/ent_agenda.html', context={'form': form,
    'username':username})
5.
6. @login_required
7. def ver_entrada_agenda(request,username):
8.     if Utente.objects.filter(numutente=request.GET['numutente']).exists():
9.         utente = Utente.objects.get(numutente=request.GET['numutente'])
10.
11.         if Entrada_Agenda.objects.filter(utente=utente).exists():
12.             entradas_agenda = Entrada_Agenda.objects.filter(utente=utente)
13.
14.             return render(request, 'funcionario/ver_ent_agenda.html',
    context={'consultas': entradas_agenda, 'username':username})
15.         else:
16.             string = "Utente inserido não tem consultas marcadas!"
17.             return render(request, "medico/erros2.html", {'string': string})
18.     else:
19.         string = "Utente inserido não existe! Insira de novo..."
20.         return render(request, "medico/erros2.html", {'string': string})

```

Um funcionário pode também ver quais os utentes que usam ou usaram um determinado medicamento. Para tal, recorre-se à *view* “medicamento” que retorna uma página onde é necessário submeter o nome do medicamento. De seguida, com a implementação da *view* “submeter_medicamento”, irá buscar o nome desse medicamento e irá buscar todos os

diferentes tipos de medicamentos com esse nome e para cada um dele buscar as prescrições onde estão. Depois para cada prescrição irá buscar as consultas onde foram receitadas. Para cada consulta irá buscar a ficha médica dessa consulta e, conseqüentemente, o utente dessa ficha médica e coloca numa lista o nome desse utente e o seu número de utente.

```

1. @login_required
2. def medicamento(request, username):
3.     form = MedicamentoNomeForm()
4.     return render(request, 'funcionario/medicamento.html', context={'form': form,
5.     'username':username})
6.
7. @login_required
8. def submeter_medicamento(request, username):
9.
10.     lista_u = []
11.     lista_con = []
12.     nome = request.GET["nome"]
13.     if Medicamento.objects.filter(nome = nome).exists():
14.         medicamento = Medicamento.objects.filter(nome = nome)
15.         for medc in medicamento:
16.             prescricao = Prescricao.objects.filter(med = medc)
17.             for el in prescricao:
18.                 consultas = el.consulta
19.                 lista_con.append(consultas)
20.
21.             for consulta in lista_con:
22.                 fichamedica = consulta.fichamedica
23.                 utente = fichamedica.utente
24.                 numutente = utente.numutente
25.                 utente_nome = utente.nome
26.
27.                 if [numutente,utente_nome] not in lista_u:
28.                     lista_u.append([numutente,utente_nome])
29.
30.             return render(request,"funcionario/submeter_medicamento.html",
31.             context={'utentes' : lista_u, 'username':username})
32.         else:
33.             string = "Medicamento inserido não existe! Tente de novo..."
34.             return render(request, "medico/erros2.html", {'string': string})

```

Por fim, o funcionário pode também ver todas as informações de um determinado utente. Recorre-se à *view* “ver_utente”, a qual retorna uma página onde é necessário colocar um número de utente. De seguida, com a *view* “ver_info_utente” irá buscar esse número de utente e a partir dele irá buscar o nome do utente. No fim, aparece uma página com as informações desse utente.

```

1. @login_required
2. def ver_utente(request, username):
3.     form = HistoricoConsultasForm()
4.     return render(request, "funcionario/numutente.html", context={'form': form,
5.     'username':username})
6.
7. @login_required
8. def ver_info_utente(request, username):
9.     numutente = request.GET["numutente"]

```

```

9.
10.     if Utente.objects.filter(numutente = numutente).exists():
11.
12.         utente = Utente.objects.get(numutente = numutente)
13.         return render(request, "funcionario/lista_utentes.html", context={'utente' :
    utente, 'username':username})
14.
15.     else:
16.         string = "Utente inserido não existe! Insira de novo..."
17.         return render(request, "medico/erros2.html", {'string': string})

```

3.3.4. Gestor

O gestor é outro utilizador da nossa aplicação que pode adicionar funcionários, adicionar médicos e medicamentos, ver lista de utentes, ver uma lista de todos os médicos, ver uma lista de todos os funcionários, ver uma lista de todos os medicamentos e ver os utentes que nasceram num determinado ano. Além disto, pode povoar, através de um ficheiro, a base de dados.

```

1. @login_required
2. def gestor(request, username):
3.     return render(request, 'gestor/gestor.html', {'username': username})

```

Para adicionar um funcionário recorre-se à *view* “adicionar_f” que retornará uma página onde será necessário submeter todas as informações de um funcionário. Depois a partir da *view* “submeter_f” vai-se adicionar um funcionário à base de dados com as informações submetidas.

```

1. @login_required
2. def adicionar_f(request, username):
3.     form = FuncionarioForm()
4.     return render(request, "gestor/adicionar_funcionario.html", context={'form': form,
    "username":username})
5.
6. @login_required
7. def submeter_f(request, username):
8.     nome = request.POST["nome"]
9.     morada = request.POST["morada"]
10.    nif = request.POST["nif"]
11.    cc = request.POST["cc"]
12.    datanasc = request.POST["datanasc"]
13.    numfunc = request.POST["numfunc"]
14.
15.    funcionario = Funcionario.objects.create(nome = nome, morada = morada, nif = nif, cc
    = cc, datanasc = datanasc, numfunc = numfunc)
16.    funcionario.save()
17.    string = "Funcionário adicionado!"
18.    return render (request, "gestor/erro.html", context={'string':string,
    "username":username})

```


Para adicionar um médico recorre-se à *view* “adicionar_m” que retornará uma página onde será necessário submeter todas as informações de um médico. Depois a partir da *view* “submeter_m” adiciona-se um médico à base de dados com as informações submetidas.

```

1. @login_required
2. def adicionar_m(request, username):
3.     form = MedicoForm()
4.     return render(request, "gestor/adicionar_medico.html", context={'form': form,
        "username":username})
5.
6. @login_required
7. def submeter_m(request, username):
8.     nome = request.POST["nome"]
9.     morada = request.POST["morada"]
10.    nif = request.POST["nif"]
11.    cc = request.POST["cc"]
12.    datanasc = request.POST["datanasc"]
13.    cedula = request.POST["cedula"]
14.    especialidade = request.POST["especialidade"]
15.    medico = Medico.objects.create(nome = nome, morada = morada, nif = nif, cc = cc,
        datanasc = datanasc, cedula = cedula, especialidade = especialidade)
16.    medico.save()
17.    string = "Médico adicionado!"
18.    return render(request, "gestor/erro.html", context={'string':string,
        "username":username})

```

Para adicionar um medicamento recorre-se à *view* “adicionar_medc” que retornará uma página onde será necessário submeter todas as informações de um medicamento. Depois a partir da *view* “submeter_medc” vai-se adicionar um medicamento à base de dados com as informações submetidas.

```

1. @login_required
2. def adicionar_medc(request, username):
3.     form = MedicamentoForm()
4.     return render(request, "gestor/adicionar_medicamento.html", context={'form': form,
        "username":username})
5.
6. @login_required
7. def submeter_medc(request, username):
8.     dci = request.POST["dci"]
9.     nome = request.POST["nome"]
10.    formafarmaceutica = request.POST["formafarmaceutica"]
11.    dosagem = request.POST["dosagem"]
12.    estadoautorizacao = request.POST["estadoautorizacao"]
13.    generico = request.POST["generico"]
14.    titular_aim = request.POST["titular_aim"]
15.    if generico == "on":
16.        generico = True
17.    else:
18.        generico = False
19.    medicamento = Medicamento.objects.create(dci = dci, nome = nome, formafarmaceutica =
        formafarmaceutica, dosagem = dosagem, estadoautorizacao = estadoautorizacao,
        generico = generico, titular_aim = titular_aim)
20.    medicamento.save()
21.    string = "Medicamento adicionado!"
22.    return render(request, "gestor/erro.html", context={'string':string,
        "username":username})

```

O gestor pode também ver a lista de utentes da base de dados. Para tal recorre-se à view “ver_utente_gestor” que irá retornar uma página com a lista de utentes.

```
1. @login_required
2. def ver_utente_gestor(request, username):
3.     utente = Utente.objects.all()
4.     return render(request, "gestor/lista_utentes.html",
       context={'utente': utente, "username":username})
```

O gestor também pode ver a lista de todos os médicos. Para tal, recorre-se à view “ver_medico” que retorna uma página as informações de todos os médicos.

```
1. @login_required
2. def ver_medico(request, username):
3.     medicos = Medico.objects.all()
4.     return render(request, 'gestor/lista_medicos.html',
       context={'medicos': medicos, "username":username})
```

O gestor também pode ver a lista de todos os funcionários. Para tal, recorre-se à view “ver_funcionario” que retorna uma página as informações de todos os funcionários.

```
1. @login_required
2. def ver_funcionario(request, username):
3.     funcionarios = Funcionario.objects.all()
4.     return render(request, 'gestor/lista_funcionarios.html',
       context={'funcionarios': funcionarios, "username":username})
```

O gestor também consegue ver a lista de todos os medicamentos. Para isso acontece, recorre-se à view “ver_medicamento”, que retorna uma página com todos os medicamentos.

```
1. @login_required
2. def ver_medicamento(request, username):
3.     medicamento = Medicamento.objects.all()
4.     return render(request, 'gestor/lista_medicamentos.html',
       context={'medicamento': medicamento, "username":username})
```

O gestor pode também ver utentes que nasceram num determinado ano. Para tal recorre-se à view “ver_idade”, a qual vai retornar uma página onde é necessário colocar o ano pretendido. De seguida, com a view “ver_idade_submeter” irá verificar para cada utente se o seu ano de nascimento é igual ao ano submetido e coloca esse utente numa lista. No fim retorna uma página onde estão o número de utente, data de nascimento e o nome de cada utente.

```

1. @login_required
2. def ver_idade(request, username):
3.     form = UtenteIdadeForm()
4.     return render(request, 'gestor/ver_utentes_idade.html', context={'form': form,
        "username":username})
5.
6. @login_required
7. def ver_idade_submeter(request, username):
8.     ano = request.GET["ano"]
9.     utentes = Utente.objects.all()
10.    lista_u = []
11.    for utente in utentes:
12.        if int(utente.datanasc.year) == int(ano):
13.            lista_u.append(utente)
14.    if len(lista_u)==0:
15.        string = "Não há ninguém com a idade pretendida! Adicione de novo..."
16.        return render(request, "gestor/erros2.html", {'string': string})
17.    else:
18.        return render(request, 'gestor/utentes_idade.html',
        context={'utentes': lista_u, "username":username})

```

3.3.5. Médico

O médico é outro utilizador da nossa aplicação que pode realizar consulta. Ao se realizar, pode marcar um exame, adicionar medições, observações, prescrições e histórico de um utente, pode também ver a ficha médica, exames e consultas do utente que está a consultar e, por fim, ver que prescrições e respetivos medicamentos foram receitados nessas mesmas consultas.

Além disto, o médico pode também ver as consultas que ele próprio realizou.

```

1. def medico(request, username):
2.     return render(request, 'medico/medico.html', {'username': username})

```

O médico pode realizar uma consulta. Para isto, pode verificar as consultas que tem marcadas num determinado dia.

```

1. @login_required
2. def entrada_agenda_medico(request,username):
3.     form = AgendaMedicoForm()
4.     return render(request, 'medico/ent_agenda.html',
        context={'form': form, 'username':username})
5.
6. @login_required
7. def ver_entrada_agenda_medico(request,username):
8.     listac=[]
9.     x = username.replace("M", "")
10.    data=request.GET['data']
11.
12.    medico = Medico.objects.get(cedula=x)
13.    if Entrada_Agenda.objects.filter(medico=medico).exists():

```

```

14.     entradas_agenda = Entrada_Agenda.objects.filter(medico=medico)
15.     for eag in entradas_agenda:
16.         if str(eag.data_hora.date()) == data:
17.             listac.append(eag)
18.         if len(listac)>0:
19.             return render(request, 'medico/ver_ent_agenda.html',
context={'consultas': listac, 'username':username})
20.         else:
21.             string = "Não tem consultas marcadas para essa data!"
22.             return render(request, "medico/erros2.html", {'string': string})
23.     else:
24.         string = "Médico inserido não tem consultas marcadas!"
25.         return render(request, "medico/erros2.html", {'string': string})

```

Para realizar uma consulta terá de submeter o número de utente que está a ter essa consulta, a data e hora, bem como a sua cédula. Se existir alguma entrada na tabela “Entrada_Agenda” com o utente desse número de utente, com essa data e hora e um médico com essa cédula. Se assim for, então irá buscar a ficha médica desse utente e irá criar uma consulta com essa data, médico e ficha médica. De seguida, essa entrada na “Entrada_Agenda” é eliminada e retorna uma página com vários botões.

```

1. @login_required
2. def realizar_consulta_adicionar(request, username):
3.     x = username.replace("M", "")
4.     if str(x) == request.POST['cedula']:
5.         utente = Utente.objects.get(numutente=request.POST['numutente'])
6.         data = request.POST['data']
7.         hora = request.POST['hora']
8.         cedula = request.POST['cedula']
9.         entradas_agenda = Entrada_Agenda.objects.all()
10.
11.         data1 = data.split("-")
12.         hora1 = hora.split(":")
13.         dt = date(int(data1[0]), int(data1[1]), int(data1[2]))
14.         tm = time(int(hora1[0]), int(hora1[1]), int(hora1[2]))
15.
16.         for eag in entradas_agenda:
17.             if utente == eag.utente and dt == eag.data_hora.date() and tm ==
eag.data_hora.time() and cedula == eag.medico.cedula:
18.                 fichamedica = Ficha_medica.objects.get(utente=utente)
19.                 medico = eag.medico
20.                 consulta = Consulta.objects.create(data= dt, medico = medico,
fichamedica = fichamedica)
21.                 consulta.save()
22.                 id = consulta.id_consulta
23.                 eag.delete()
24.                 idu=utente.numutente
25.                 return render(request, 'medico/consulta.html',
context={'id':id, "username":username, 'idu':idu})
26.
27.                 string= "Não é possível realizar esta consulta... Retifique!"
28.                 return render(request, "medico/erro.html", context={'string':string,
"username":username})
29.             else:
30.                 string = "O número introduzido não é o seu... Retifique!"
31.                 return render(request, "medico/erros2.html", {'string': string})

```

Ao realizar uma consulta, o médico pode marcar um exame. Para tal, recorre-se à *view* “marcar_exame” que retorna uma página onde é necessário submeter as informações precisas para marcar um exame. De seguida, na *view* “submeter_exame”, o servidor irá buscar todas as informações anteriormente submetidas e irá criar uma entrada na tabela “Exame” com essas informações.

```

1. @login_required
2. def marcar_exame(request, username, id, idu):
3.     form = MarcarExameForm()
4.     id = id
5.     return render(request, 'medico/marcar_exame.html', context={'form': form, 'id':id,
6.         "username":username, 'idu':idu})
7. @login_required
8. def submeter_exame(request, username, id, idu):
9.     tipo = request.POST['tipo']
10.    local = request.POST['local']
11.    data_hora = request.POST['data_hora']
12.    duracao = request.POST['duracao']
13.    preco = request.POST['preco']
14.    estado = request.POST['estado']
15.    observacoes = request.POST['observacoes']
16.    if estado == "on":
17.        estado = True
18.    else:
19.        estado = False
20.    consulta = Consulta.objects.get(id_consulta= id)
21.    exame = Exame.objects.create(tipo = tipo, local = local, data_hora = data_hora,
22.        duracao = duracao, preco = preco, estado = estado, observacoes = observacoes,
23.        consulta = consulta)
24.    exame.save()
25.    string = "Exame criado!"
26.    return render(request, "medico/erros2.html", {'string':string})

```

O médico quando realiza uma consulta pode também adicionar medições. Para tal, recorre-se à *view* “adicionar_medicoes” que irá retornar uma página com as informações necessárias para poder criar uma medição. De seguida, na *view* “submeter_medicoes”, o servidor irá buscar todas as informações anteriormente submetidas e vai criar uma entrada na tabela “Medicao” com essas informações e irá adicionar essa medição à ficha médica do utente.

```

1. @login_required
2. def adicionar_medicoes(request, username, id, idu):
3.     form = MedicoesForm()
4.     return render(request, 'medico/medicoes.html', context={'form': form, 'id':id,
5.         "username":username, 'idu':idu})
6. @login_required
7. def submeter_medicoes(request, username, id, idu):
8.     numutente = request.POST["numutente"]
9.     data = request.POST["data"]
10.    peso = request.POST["peso"]
11.    glicemia = request.POST["glicemia"]
12.    altura = request.POST["altura"]
13.    tensaoarterial = request.POST["tensaoarterial"]

```

```

14.     colesterol = request.POST["colesterol"]
15.     trigliceridios = request.POST["trigliceridios"]
16.     saturacao = request.POST["saturacao"]
17.     inr = request.POST["inr"]
18.     if idu == numutente:
19.         utente = Utente.objects.get(numutente = numutente)
20.         fichamedica= Ficha_medica.objects.get(utente = utente)
21.         medicao = Medicao.objects.create(data = data, peso = peso, glicemia= glicemia,
altura = altura, tensaoarterial = tensaoarterial,colesterol = colesterol, trigliceridios
= trigliceridios, saturacao = saturacao, inr = inr)
22.         medicao.save()
23.         fichamedica.medicoes.add(medicao)
24.
25.         string="Medição adicionada!"
26.         return render(request,"medico/erros2.html",{ 'string':string})
27.     else:
28.         string = "Está a submeter medições ao utente errado... Verifique!"
29.         return render(request, "medico/erros2.html", { 'string': string})

```

Um médico, ao realizar uma consulta, pode também ver informações de um determinado medicamento, caso queria adicionar posteriormente uma prescrição com um medicamento. Para tal, recorre-se à *view* “infomedicamento” que irá retornar uma página que contém uma caixa para submeter um nome de um medicamento. De seguida, com a *view* “verinfomedicamento”, o servidor irá buscar esse nome e todos os medicamentos com esse nome. Por fim, retorna uma página com informações desses medicamentos.

```

1. @login_required
2. def infomedicamento(request, username, id, idu):
3.     form = MedicamentoNomeForm()
4.     return render(request, 'medico/medicamento_submeter.html', context={'form': form,
'id':id, "username":username, 'idu':idu})
5.
6. @login_required
7. def verinfomedicamento(request, username, id, idu):
8.     nome = request.GET["nome"]
9.     if Medicamento.objects.filter(nome=nome).exists():
10.         medicamento = Medicamento.objects.filter(nome=nome)
11.         return render(request, 'medico/lista_medicamento.html', context={'medicamento':
medicamento, 'id':id, "username":username, 'idu':idu})
12.     else:
13.         string = "Medicamento introduzido não existe! Adicione de novo..."
14.         return render(request, "medico/erros2.html", { 'string': string})

```

O médico quando realiza uma consulta pode também adicionar prescrições. Para tal, recorre-se à *view* “adicionar_prescrições” que irá retornar uma página com as informações necessárias para poder criar uma prescrição. De seguida, na *view* “ submeter_prescrições”, o servidor irá buscar todas as informações anteriormente submetidas e vai criar uma entrada na tabela “Prescricao” com essas informações e irá adicionar essa prescricao à consulta que está a ser realizada.

```

1. @login_required
2. def adicionar_prescricoes(request, username,id, idu):
3.     form = PrescricoesForm()
4.     return render(request, 'medico/adicionar_prescricoes.html', context={'form': form,
5.     'id':id, "username":username, 'idu':idu})
6.
7. @login_required
8. def submeter_prescricoes(request, username,id, idu):
9.     data = request.POST["data"]
10.    medicamento = request.POST["medicamento"]
11.    dosagem = request.POST["dosagem"]
12.    toma = request.POST["toma"]
13.    if ", " in medicamento:
14.        consulta = Consulta.objects.get(id_consulta=id)
15.        prescricao = Prescricao.objects.create(data=data, toma=toma)
16.        prescricao.consulta = consulta
17.        prescricao.save()
18.        medicamento1 = medicamento.split(",")
19.        dosagem1 = dosagem.split(",")
20.        for i in range(len(medicamento1)):
21.            medc = Medicamento.objects.get(nome=medicamento1[i], dosagem=dosagem1[i])
22.            prescricao.med.add(medc)
23.            string = "Prescrição adicionada!"
24.            return render(request, "medico/erros2.html", {'string': string})
25.    else:
26.        medc = Medicamento.objects.get(nome=medicamento, dosagem=dosagem)
27.        consulta = Consulta.objects.get(id_consulta=id)
28.        prescricao = Prescricao.objects.create(data=data, toma=toma)
29.        prescricao.consulta = consulta
30.        prescricao.save()
31.        prescricao.med.add(medc)
32.        string = "Prescrição adicionada!"
33.        return render(request, "medico/erros2.html", {'string': string})

```

O médico quando realiza uma consulta pode também adicionar observações. Para tal, recorre-se à view “adicionar_observacoes” que irá retornar uma página com uma caixinha para poder criar uma observação. De seguida, na view “submeter_observacoes”, o servidor irá buscar a consulta que está a ser realizada e irá adicionar observações a essa consulta.

```

1. @login_required
2. def adicionar_observacoes(request, username,id, idu):
3.     form = ObservacoesForm()
4.     return render(request, 'medico/adicionar_observacoes.html', context={'form': form,
5.     'id':id, "username":username, 'idu':idu})
6.
7. @login_required
8. def submeter_observacoes(request, username,id, idu):
9.     observacoes = request.POST["observacoes"]
10.    consulta = Consulta.objects.get(id_consulta= id)
11.    consulta.observacoes = observacoes
12.    consulta.save()
13.    string = "Observação adicionada!"
14.    return render(request, "medico/erros2.html", {'string':string})

```

O médico quando realiza uma consulta pode também adicionar histórico. Para tal, recorre-se à view “adicionar_historico” que irá retornar uma página com uma caixinha para

poder adicionar histórico e outra para submeter o número do utente. De seguida, na view “submeter_historico”, o servidor irá buscar a ficha médica a partir do utente com o número do utente submetido e irá adicionar o histórico submetido a essa ficha médica .

```

1. @login_required
2. def adicionar_historico(request, username,id, idu):
3.     form = HistoricoForm()
4.     id = id
5.     return render(request, 'medico/adicionar_historico.html', context={'form': form,
        'id':id, "username":username, 'idu':idu})
6.
7. @login_required
8. def submeter_historico(request, username,id, idu):
9.     numutente = request.POST["numutente"]
10.    historico = request.POST["historico"]
11.    if idu == numutente:
12.        utente = Utente.objects.get(numutente=numutente)
13.        fichamedica = Ficha_medica.objects.get(utente=utente)
14.        fichamedica.historico += historico
15.        fichamedica.save()
16.        string = "Histórico adicionado à Ficha Médica!"
17.        return render(request, "medico/erros2.html", {'string': string})
18.    else:
19.        string = "Está a submeter histórico ao utente errado... Verifique!"
20.        return render(request, "medico/erros2.html", {'string': string})

```

Além de todas as ações anteriores, o médico durante a consulta pode também ver a ficha médica do utente que está a consultar. Para isto recorre-se à view “ver_ficha_u”, que seleciona a ficha médica desse utente. Irá, depois, buscar todas medições contidas nessa ficha médica e retorna uma página onde aparecem as informações da ficha médica e das medições.

```

1. @login_required
2. def ver_ficha_u(request, username,id, idu):
3.     utente = Utente.objects.get(numutente=idu)
4.     fichamedica = Ficha_medica.objects.get(utente=utente)
5.     if Medicao.objects.all().exists():
6.         medicoes = fichamedica.medicoes.all()
7.         return render(request, "medico/fichautente.html",
            context={'fichamedica': fichamedica, 'medicoes': medicoes, "username":
            username, 'id':id, 'idu': idu})
8.     else:
9.         return render(request, "medico/fichautente.html", context={
            'fichamedica': fichamedica, "username": username, 'id':id, 'idu': idu})

```

O médico pode também ver os exames do utente que consulta. Recorre-se à view “ver_exame_u” que irá buscar a sua ficha médica e as consultas contidas nessa ficha médica. Depois, para cada consulta irá buscar os seus exames e adiciona-os a uma lista. No fim, retorna uma página onde aparecem as informações desses exames.


```

1. @login_required
2. def ver_exame_u(request, username,id, idu):
3.     lista_examenes = []
4.     utente = Utente.objects.get(numutente=idu)
5.     fichamedica = Ficha_medica.objects.get(utente=utente)
6.     consultas = Consulta.objects.filter(fichamedica=fichamedica)
7.     for consulta in consultas:
8.         exames = Exame.objects.filter(consulta=consulta)
9.         if exames:
10.             lista_examenes.append(exames)
11.     if len(lista_examenes) == 0:
12.         string = "O Utente inserido não tem Exames!"
13.         return render(request, "medico/erros2.html", {'string': string})
14.     else:
15.         return render(request, "medico/exames_utente.html", context={'exames':
            lista_examenes, "username": username, 'id':id, 'idu': idu})

```

O médico pode ver o histórico de consultas do utente que consulta. Recorre-se à view “ver_consulta_u” que irá buscar a sua ficha médica e as consultas contidas nessa ficha médica. No fim, retorna uma página onde aparecem as informações dessas consultas.

```

1. @login_required
2. def ver_consulta_u(request, username,id, idu):
3.     utente = Utente.objects.get(numutente=idu)
4.     fichamedica = Ficha_medica.objects.get(utente=utente)
5.     if Consulta.objects.filter(fichamedica=fichamedica).exists():
6.         consultas = Consulta.objects.filter(fichamedica=fichamedica)
7.         return render(request, "medico/consultas_utente.html",
            context={'consultas': consultas, "username": username, 'id':id, 'idu': idu})
8.     else:
9.         string = "Utente inserido não tem consultas realizadas!"
10.        return render(request, "medico/erros2.html", {'string': string})

```

O médico pode ver as consultas de um determinado utente. Recorre-se à view “ver_prescrições” que retorna uma página onde aparece uma entrada para submeter um id de uma consulta. Esta consulta tem de pertencer ao utente que está a ser consultado. De seguida, com a view “prescrições”, o servidor irá verificar se a consulta com esse id existe e se sim, então irá buscar as prescrições dessa consulta. No fim, retorna uma página onde aparecem as informações dessas prescrições.

```

1. @login_required
2. def ver_prescricoes(request, username,id, idu):
3.     form = ConsultaForm()
4.     return render(request, "medico/ver_prescricoes_c.html", context={'form': form,
        "username": username, 'id':id, 'idu': idu})
5.
6. @login_required
7. def prescricoes(request, username,id, idu):
8.     form = PrescricaoForm()
9.     id = request.GET["id_con"]
10.    lista_id = ""
11.    utente=Utente.objects.get(numutente=idu)

```

```

12.     fichamedica=Ficha_medica.objects.get(utente=utente)
13.     if Consulta.objects.filter(id_consulta=id, fichamedica=fichamedica).exists():
14.         consulta = Consulta.objects.get(id_consulta=id, fichamedica=fichamedica)
15.         if Prescricao.objects.filter(consulta=consulta).exists():
16.             prescicoes = Prescricao.objects.filter(consulta=consulta)
17.             for presc in prescicoes:
18.                 lista_id += "_" + str(presc.id_prescricao)
19.             return render(request, "medico/prescicoes.html", context={'prescicoes':
prescicoes, 'form': form, "username": username, 'lista_id': lista_id, 'id':id,
'idu': idu})
20.         else:
21.             string = "Não há Prescrições associadas à Consulta pretendida!"
22.             return render(request, "medico/erros2.html", {'string': string})
23.     else:
24.         string = "Consulta inserida não existe ou não pertence ao Utente! Verifique as c
onsultas do Utente..."
25.         return render(request, "medico/erros2.html", {'string': string})

```

Com a seguinte *view*, o médico depois de obter as prescrições de uma consulta poderá colocar um id de uma prescrição dessa mesma consulta e verificar para essa prescrição, os medicamentos contidos. No fim, retorna uma página que contém as informações desses medicamentos.

```

1. @login_required
2. def medicamentos(request, username, id, lista_id, idu):
3.     id = request.GET["id_presc"]
4.     lista = lista_id.split("_")
5.     if id in lista_id:
6.         prescricao = Prescricao.objects.get(id_prescricao=id)
7.         medc = prescricao.med.all()
8.         return render(request, "medico/medicamentos.html", context={'medicamentos':
medc, "username": username, 'id':id, 'idu': idu})
9.     else:
10.         string = "Esta prescrição não pertence à consulta pretendida!"
11.         return render(request, "medico/erros2.html", {'string': string})

```

Por último, e sem ter de realizar qualquer consulta, o médico pode também ver as suas consultas e o utente para o qual essa consulta foi feita. Recorrendo à *view* “histórico_consultas_medico”, o servidor irá buscar todas as consultas do médico. Depois, para cada consulta irá buscar a ficha médica e, por sua vez, o utente dessa ficha e adiciona para uma lista esse utente e consulta e, por fim, adiciona essa lista a uma lista final. Acabamos por ter uma lista de listas em que cada sublista contém a consulta e o seu utente.

```

1. @login_required
2. def historico_consultas_medico(request, username):
3.     x=username.replace("M", "")
4.     medico = Medico.objects.get(cedula=x)
5.     if Consulta.objects.filter(medico=medico).exists():
6.         consultas = Consulta.objects.filter(medico=medico)
7.         listas_utente_consulta=[]
8.         for consulta in consultas:

```

```

9.         lista_utente_consulta = []
10.        fichamedica = consulta.fichamedica
11.        utente = fichamedica.utente
12.        lista_utente_consulta.append(utente)
13.        lista_utente_consulta.append(consulta)
14.        listas_utente_consulta.append(lista_utente_consulta)
15.        return render(request, 'medico/ver_historico_consultas.html', context={'consulta
s':listas_utente_consulta, "username":username})
16.    else:
17.        string = "Não existem consultas associadas ao médico pretendido!"
18.        return render(request, "medico/erros2.html", {'string': string})

```

3.4. Povoamento

De modo a povoar a base de dados criada com os dados previamente fornecidos pelo docente, foram criadas *views* que vão buscar para cada linha do ficheiro fornecido todas as informações necessárias para adicionar o pretendido. Esta tarefa é feita apenas pelo Gestor.

```

1. @login_required
2. def povoar(request, username):
3.     return render(request, 'gestor/povoargeral.html', context={"username": username})

```

Todo o código referente a estes povoamentos encontra-se em anexo.

3.5. Templates

Uma vez que o Django é uma estrutura Web, este precisa de uma maneira conveniente para gerar HTML dinamicamente. A abordagem mais comum baseia-se na criação de templates. Um template contém as partes estáticas da saída HTML desejada, bem como algumas sintaxes especiais que descrevem como o conteúdo dinâmico será inserido. É apresentado, como exemplo, o código HTML do template da página principal do projeto e o resultado na página WEB, implementado na figura 2.

```

1. {% extends 'base.html' %}
2. {% block content %}
3.     <form method="post">
4.         {% csrf_token %}
5.
6.         <button class="w3-button w3-block w3-black" style="margin-left:auto;margin-
right:auto;display:block;margin-top:0%;margin-
bottom:0%;width:40%"><a href="/alves_lda/logingestor/" class="w3-button w3-block w3-
black">GESTOR</a></button>
7.
8.         <button class="w3-button w3-block w3-black" style="margin-left:auto;margin-
right:auto;display:block;margin-top:0%;margin-

```

```

9. bottom:0%;width:40%"><a href="/alves_lda/loginmedico/" class="w3-button w3-block w3-
10. black">MÉDICO</a></button>
11.
12. <button class="w3-button w3-block w3-black" style="margin-left:auto;margin-
13. right:auto;display:block;margin-top:0%;margin-
14. bottom:0%;width:40%"><a href="/alves_lda/loginfuncionario/" class="w3-button w3-
15. block w3-black">FUNCIONÁRIO</a></button>
16.
17. <button class="w3-button w3-block w3-black" style="margin-left:auto;margin-
18. right:auto;display:block;margin-top:0%;margin-
19. bottom:0%;width:40%"><a href="/alves_lda/loginutente/" class="w3-button w3-block w3-
20. black">UTENTE</a></button>
21.
22. </form>
23. {% endblock %}

```



Figura 2 - Página Inicial

3.6. URL's

No arquivo *urls.py* são inseridos os caminhos que transformam cada uma das expressões simplificadas em caminhos da URL para as funções que as executam. O URL corresponde ao endereço digitado no navegador que executará uma ação específica.

```

1. from django.contrib import admin
2. from django.urls import path
3. from . import views
4.
5. app_name = "alves_lda"
6.
7. urlpatterns = [
8.
9. #-----INICIO-----#
10.
11. path('admin/', admin.site.urls),
12. path('', views.homepage_view, name="inicial"),
13. path('sair/', views.sair),
14.
15. #-----UTENTE-----#

```

```

16.
17.     path('loginutente/', views.login_utente),
18.     path('registar_utente/', views.registar_utente),
19.     path('utente/<str:username>', views.utente),
20.     path('utente/<str:username>/marcar_consulta/', views.marcar_consulta),
21.     path('utente/<str:username>/marcar_consulta/marcar/', views.marcar),
22.     path('utente/<str:username>/marcar_consulta/horariofuncionamento/', views.horario),
23.     path('utente/<str:username>/marcar_consulta/especialidades', views.especialidades),
24.     path('utente/<str:username>/historico_consultas/', views.historico_consultas),
25.     path('utente/<str:username>/historico_exames/', views.historico_exames),
26.     path('utente/<str:username>/veragenda/', views.veragenda),
27.     path('utente/<str:username>/cancelar_consulta/', views.cancelar_consulta),
28.     path('utente/<str:username>/cancelar_consulta/cancelar/', views.cancelar),
29.
30. #-----MEDICO-----#
31.
32.     path('loginmedico/', views.login_medico),
33.     path('registar_medico/', views.registar_medico),
34.     path('medico/<str:username>', views.medico),
35.     path('medico/<str:username>/hist_consultas/', views.historico_consultas_medico),
36.     path('medico/<str:username>/realizar_consulta/', views.realizar_consulta),
37.     path('medico/<str:username>/realizar_consulta/entradagenda_medico/',
38.          views.entrada_agenda_medico),
39.     path('medico/<str:username>/realizar_consulta/entradagenda_medico/ver',
40.          views.ver_entrada_agenda_medico),
41.     path('medico/<str:username>/realizar_consulta/adicionar/',
42.          views.realizar_consulta_adicionar),
43.     path('medico/<str:username>/realizar_consulta/adicionar/marcar_exame/<int:id>/
44.          <str:idu>', views.marcar_exame),
45.     path('medico/<str:username>/realizar_consulta/adicionar/marcar_exame/submeter/
46.          <int:id>/<str:idu>', views.submeter_exame),
47.     path('medico/<str:username>/realizar_consulta/adicionar/medicoes/<int:id>/
48.          <str:idu>', views.adicionar_medicoes),
49.     path('medico/<str:username>/realizar_consulta/adicionar/medicoes/submeter/
50.          <int:id>/<str:idu>', views.submeter_medicoes),
51.     path('medico/<str:username>/realizar_consulta/adicionar/prescricoes/<int:id>/
52.          <str:idu>', views.adicionar_prescricoes),
53.     path('medico/<str:username>/realizar_consulta/adicionar/prescricoes/submeter/
54.          <int:id>/<str:idu>', views.submeter_prescricoes),
55.     path('medico/<str:username>/realizar_consulta/adicionar/observacoes/<int:id>/
56.          <str:idu>', views.adicionar_observacoes),
57.     path('medico/<str:username>/realizar_consulta/adicionar/observacoes/submeter/
58.          <int:id>/<str:idu>', views.submeter_observacoes),
59.     path('medico/<str:username>/realizar_consulta/adicionar/historico/<int:id>/
60.          <str:idu>', views.adicionar_historico),
61.     path('medico/<str:username>/realizar_consulta/adicionar/historico/submeter/
62.          <int:id>/<str:idu>', views.submeter_historico),
63.     path('medico/<str:username>/realizar_consulta/adicionar/medicamento/<int:id>/
64.          <str:idu>', views.infomedicamento),
65.     path('medico/<str:username>/realizar_consulta/adicionar/medicamento/ver/<int:id>/<st
66.          r:idu>', views.verinfomedicamento),
67.     path('medico/<str:username>/realizar_consulta/adicionar/ver_ficha_u/<int:id>/
68.          <str:idu>', views.ver_ficha_u),
69.     path('medico/<str:username>/realizar_consulta/adicionar/ver_exames_u/<int:id>/
70.          <str:idu>', views.ver_exame_u),
71.     path('medico/<str:username>/realizar_consulta/adicionar/ver_consultas_u/<int:id>/
72.          <str:idu>', views.ver_consulta_u),
73.     path('medico/<str:username>/realizar_consulta/adicionar/ver_prescricoes_c/<int:id>/
74.          <str:idu>', views.ver_prescricoes),
75.     path('medico/<str:username>/realizar_consulta/adicionar/ver_prescricoes_c/
76.          prescricoes/<int:id>/<str:idu>', views.prescricoes),
77.     path('medico/<str:username>/realizar_consulta/adicionar/ver_prescricoes_c/
78.          prescricoes/<str:lista_id>/medicamento/<int:id>/<str:idu>', views.medicamentos),
79.
80. #-----FUNCIONÁRIO-----#

```

```

61.
62.     path('loginfuncionario/', views.login_funcionario),
63.     path('registar_funcionario/', views.registar_funcionario),
64.     path('funcionario/<str:username>', views.funcionario),
65.     path('funcionario/<str:username>/adicionar_utente/', views.adicionar_u),
66.     path('funcionario/<str:username>/adicionar_utente/submeter', views.submeter_u),
67.     path('funcionario/<str:username>/entrada_agenda/', views.entrada_agenda),
68.     path('funcionario/<str:username>/entrada_agenda/ver', views.ver_entrada_agenda),
69.     path('funcionario/<str:username>/medicamento/', views.medicamento),
70.     path('funcionario/<str:username>/medicamento/submeter', views.submeter_medicamento),
71.     path('funcionario/<str:username>/ver_lista_u/', views.ver_utente),
72.     path('funcionario/<str:username>/ver_lista_u/submeter', views.ver_info_utente),
73.
74. #-----GESTOR-----#
75.
76.     path('loggingestor/', views.login_gestor),
77.     path('registar_gestor/', views.registar_gestor),
78.     path('gestor/<str:username>', views.gestor),
79.     path('gestor/<str:username>/adicionar_funcionario/', views.adicionar_f),
80.     path('gestor/<str:username>/adicionar_funcionario/submeter', views.submeter_f),
81.     path('gestor/<str:username>/adicionar_medico/', views.adicionar_m),
82.     path('gestor/<str:username>/adicionar_medico/submeter', views.submeter_m),
83.     path('gestor/<str:username>/adicionar_medicamento/', views.adicionar_medc),
84.     path('gestor/<str:username>/adicionar_medicamento/submeter', views.submeter_medc),
85.     path('gestor/<str:username>/ver_listaU/', views.ver_utente_gestor),
86.     path('gestor/<str:username>/ver_listaM/', views.ver_medico),
87.     path('gestor/<str:username>/ver_listaF/', views.ver_funcionario),
88.     path('gestor/<str:username>/ver_listaMedicamentos/', views.ver_medicamento),
89.     path('gestor/<str:username>/ver_idade/', views.ver_idade),
90.     path('gestor/<str:username>/ver_idade/submeter', views.ver_idade_submeter),
91.     path('gestor/<str:username>/povoar/', views.povoar),
92.     path('gestor/<str:username>/povoar/populate_utentes/', views.popular_utentes),
93.     path('gestor/<str:username>/povoar/populate_utentes/popular',
94.          views.populate_utentes),
95.     path('gestor/<str:username>/povoar/populate_medicos/', views.popular_medicos),
96.     path('gestor/<str:username>/povoar/populate_medicos/popular',
97.          views.populate_medicos),
98.     path('gestor/<str:username>/povoar/populate_funcionarios/',
99.          views.popular_funcionarios),
100.    path('gestor/<str:username>/povoar/populate_funcionarios/popular',
101.         views.populate_funcionarios),
102.    path('gestor/<str:username>/povoar/populate_medicamentos/',
103.         views.popular_medicamentos),
104.    path('gestor/<str:username>/povoar/populate_medicamentos/popular',
105.         views.populate_medicamentos),
106.    path('gestor/<str:username>/povoar/populate_fichasmedicas/',
107.         views.populate_fichasmedicas),
108.    path('gestor/<str:username>/povoar/add_historico/', views.popular_historico),
109.    path('gestor/<str:username>/povoar/add_historico/popular', views.add_historico),
110.    path('gestor/<str:username>/povoar/populate_consultas/', views.popular_consultas),
111.    path('gestor/<str:username>/povoar/populate_consultas/popular',
112.         views.populate_consultas),
113.
114. ]
115. #-----FIM-----#

```

3.7. ADMIN

Uma das partes mais poderosas do Django é a interface de administração automática. O Admin lê metadados para fornecer uma interface rápida e centrada no modelo, onde diferentes utilizadores confiáveis podem gerenciar o conteúdo do site. O uso da interface Admin é limitada pelo administrador.

O administrador pode adicionar, apagar ou alterar dados ou apenas visualizá-los. Um exemplo da página inicial do Admin está representado na figura 3.

```
1. from django.contrib import admin
2.
3. # Register your models here.
4. from .models import *
5. admin.site.register(Gestor)
6. admin.site.register(Medico)
7. admin.site.register(Funcionario)
8. admin.site.register(Utente)
9. admin.site.register(Medicamento)
10. admin.site.register(Prescricao)
11. admin.site.register(Medicacao)
12. admin.site.register(Consulta)
13. admin.site.register(Ficha_medica)
14. admin.site.register(Entrada_Agenda)
```

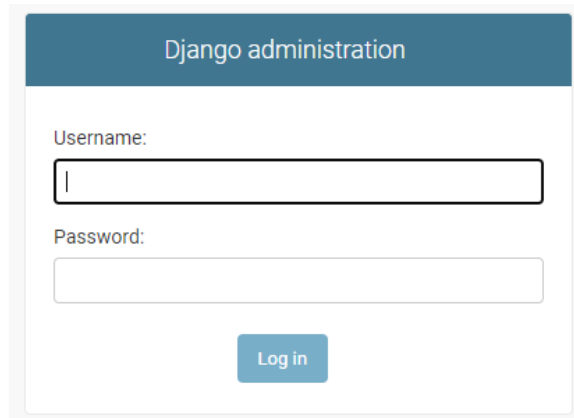


Figura 3 - Página Inicial do Admin

4. Conclusão

O projeto realizado permitiu consolidar o conhecimento relativamente à linguagem Python e, consequentemente a sua manipulação e utilização na framework pedida para este trabalho, Django's Framework.

A base de dados utilizada foi SQLITE3, a qual possibilitou a criação de uma arquitetura cliente-servidor com diferentes funcionalidades, dependendo do tipo de utilizador que está a usar a aplicação. Este sistema permite que diferentes utentes, médicos, funcionários e o gestor da clínica possam usufruir da aplicação.

A complexidade do trabalho foi notória, visto que, foi a primeira vez que foi utilizada esta framework e, por esse motivo, necessitou de uma extensa pesquisa antes de se poder começar com a implementação da aplicação. Contudo, apesar da complexidade e das eventuais dificuldades encontradas durante a efetuação do trabalho, conseguiu-se implementar um sistema capaz de abranger diferentes utilizadores, cada tipo com as suas próprias funcionalidades. Na aplicação, além de ser possível realizar pedidos para adicionar algo à base de dados é também possível realizar pedidos para obter determinados dados que se encontram na mesma. Ademais é possível um agendamento, cancelamento e a realização de consultas.

O sistema implementado cumpre com os requisitos previamente estabelecidos para o trabalho e com mais umas funcionalidades, como os logins, registos de utilizadores, bem como a realização de consultas além das suas marcações e cancelamentos.

No futuro, considera-se importante a implementação de mais algumas funcionalidades que poderão vir a ser importantes para determinadas situações, como por exemplo: diferentes durações de consultas consoante a especialidade pretendida ou até mesmo consoante o dia pretendido uma vez que, poderá haver uma maior quantidade de staff em alguns dias comparativamente a outros.

5. Bibliografia

- [1] <https://www.sqlite.org/index.html>, consultado em 20 dezembro, 2020;
- [2] <https://djangoproject.com/>, consultado em 20 dezembro, 2020;
- [3] <https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Django/Introdu%C3%A7%C3%A3o>, consultado em 20 dezembro, 2020;

6. Anexos

A. Povoamento

```

1. @login_required
2. def povoar(request, username):
3.     return render(request, 'gestor/povoageral.html', context={"username": username})
4.
5. @login_required
6. def popular_utentes(request, username):
7.     form = FicheiroForm()
8.     return render(request, 'gestor/ficheioutente.html', context={'form': form, "username":
    username})
9.
10. @login_required
11. def populate_utentes(request, username):
12.
13.     ficheiro = request.POST["ficheiro"]
14.
15.     with open(ficheiro, encoding="utf8" ) as f:
16.
17.         while True:
18.             linha = f.readline()
19.             if not linha:
20.                 break
21.             utente = linha.split(";")
22.             for i in range(len(utente)):
23.                 if utente[i].startswith("\"):
24.                     utente[i] = utente[i].replace('""', ' ').strip()
25.             lista_dn = utente[4].split("-")
26.             for i in range(len(lista_dn)):
27.                 if lista_dn[i].startswith("\"):
28.                     lista_dn[i] = lista_dn[i].replace('""', ' ').strip()
29.             lista_dn[2] = lista_dn[2].replace('""', ' ').strip()
30.             utente[8] = utente[8].replace("\n", " ").strip()
31.             utente[7] = utente[7].replace("(", " ").strip()
32.             utente[7] = utente[7].replace(")", " ").strip()
33.
34.             u = Utente.objects.create(nome = str(utente[0]),
35.                                     morada = str(utente[1]),
36.                                     nif = str(utente[2]),
37.                                     cc = str(utente[3]),
38.                                     datanasc = date(int(lista_dn[0]), int(lista_dn[1]), int
    (lista_dn[2])),
39.                                     numutente = str(utente[5]),
40.                                     telefone = str(utente[6]), telefone_emergencia = str(
    utente[7]), email = str(utente[8]))
41.             u.save()
42.
43.             string = "Utentes Povoados!"
44.             return render(request, "gestor/erro.html", {'string': string, "username": username })
45.
46. @login_required
47. def popular_medicos(request, username):
48.     form = FicheiroForm()
49.     return render(request, 'gestor/ficheiromedico.html', context={'form': form, "username":
    username})
50.
51. @login_required
52. def populate_medicos(request, username):
53.     ficheiro = request.POST["ficheiro"]
54.
55.     with open(ficheiro, encoding="utf8") as f:

```

```

56.         while True:
57.             linha = f.readline()
58.             if not linha:
59.                 break
60.             medico = linha.split(";")
61.             for i in range(len(medico)):
62.                 if medico[i].startswith("\"):
63.                     medico[i] = medico[i].replace("'", ' ').strip()
64.             lista_dn = medico[4].split("-")
65.             m = Medico.objects.create(nome=str(medico[0]),
66.                                     morada=str(medico[1]),
67.                                     nif=str(medico[2]),
68.                                     cc=str(medico[3]),
69.                                     datanasc=date(int(lista_dn[0]), int(lista_dn[1]),
70.                                     int(lista_dn[2])),
71.                                     cedula=str(medico[5]),
72.                                     especialidade=str(medico[6]))
73.             m.save()
74.             string = "Medicos Povoados!"
75.             return render(request, "gestor/erro.html", {'string': string, "username": username })
76. @login_required
77. def popular_funcionarios(request, username):
78.     form = FicheiroForm()
79.     return render(request, 'gestor/ficheirofunc.html', context={'form': form, "username":
80.     username})
81. @login_required
82. def populate_funcionarios(request, username):
83.
84.     ficheiro = request.POST["ficheiro"]
85.
86.     with open(ficheiro, encoding="utf8") as f:
87.         while True:
88.             linha = f.readline()
89.             if not linha:
90.                 break
91.             funcionario = linha.split(";")
92.             for i in range(len(funcionario)):
93.                 if funcionario[i].startswith("\"):
94.                     funcionario[i] = funcionario[i].replace("'", ' ').strip()
95.             lista_dn = funcionario[4].split("-")
96.             funcionario[5] = funcionario[5].replace("\n", " ").strip()
97.             func = Funcionario.objects.create(nome=str(funcionario[0]),
98.                                             morada=str(funcionario[1]),
99.                                             nif=str(funcionario[2]),
100.                                             cc=str(funcionario[3]),
101.                                             datanasc=date(int(lista_dn[0]), int(lista_dn[1]),
102.                                             int(lista_dn[2])),
103.                                             numfunc=int(funcionario[5]))
104.             func.save()
105.             string = "Funcionários Povoados!"
106.             return render(request, "gestor/erro.html", {'string': string, "username": username}
107.             )
108. @login_required
109. def popular_medicamentos(request, username):
110.     form = FicheiroForm()
111.     return render(request, 'gestor/ficheirosmedicamento.html', context={'form': form, "user
112.     name": username})
113. @login_required
114. def populate_medicamentos(request, username):
115.     ficheiro = request.POST["ficheiro"]
116.

```

```

117.     with open(ficheiro, encoding="utf8") as f:
118.
119.         while True:
120.             linha = f.readline()
121.             if not linha:
122.                 break
123.             medicamento = linha.split(";")
124.             if str(medicamento[6]) == "N":
125.                 medicamento[6] = False
126.             else:
127.                 medicamento[6] = True
128.             medc = Medicamento.objects.create(
129.                 dci=str(medicamento[1]),
130.                 nome=str(medicamento[2]),
131.                 formafarmaceutica=str(medicamento[3]),
132.                 dosagem=str(medicamento[4]),
133.                 estadoautorizacao=str(medicamento[5]), generico=
134.                 medicamento[6], titular_aim=str(medicamento[7]))
135.             medc.save()
136.             string = "Medicamentos Povoados!"
137.             return render(request, "gestor/erro.html", {'string': string, "username": username}
138. )
139. @login_required
140. def populate_fichasmedicas(request, username):
141.     utentes = Utente.objects.all()
142.     for utente in utentes:
143.         fichamedica = Ficha_medica(utente = utente)
144.         fichamedica.save()
145.     string = "Fichas Médicas Povoadas!"
146.     return render(request, "gestor/erro.html", {'string': string, "username": username})
147.
148. @login_required
149. def popular_historico(request, username):
150.     form = FicheiroForm()
151.     return render(request, 'gestor/ficheirohist.html', context={'form': form, "username":
152. username})
153. def add_historico(request, username):
154.     ficheiro = request.POST["ficheiro"]
155.
156.     with open(ficheiro, encoding="utf8") as f:
157.
158.         while True:
159.             linha = f.readline()
160.             if not linha:
161.                 break
162.             tokens = linha.split(";")
163.             for i in range(len(tokens)):
164.                 if tokens[i].startswith("\"):
165.                     tokens[i] = tokens[i].replace('"', ' ').strip()
166.             utente = Utente.objects.get(numutente = tokens[0])
167.             fichamedica = Ficha_medica.objects.get(utente = utente)
168.             fichamedica.historico = tokens[1]
169.             fichamedica.save()
170.             string = "Histórico Povoado!"
171.             return render(request, "gestor/erro.html", {'string': string, "username": username}
172. )
173. @login_required
174. def popular_consultas(request, username):
175.     form = FicheiroForm()
176.     return render(request, 'gestor/ficheirocons.html', context={'form': form, "username":
177. username})

```

```

178. @login_required
179. def populate_consultas(request, username):
180.     ficheiro = request.POST["ficheiro"]
181.
182.     with open(ficheiro, encoding="utf8") as f:
183.
184.         k=0
185.         while k<10001:
186.             linha = f.readline()
187.             if not linha:
188.                 break
189.             tokens = linha.split(";")
190.             for i in range(len(tokens)):
191.                 if tokens[i].startswith("\"):
192.                     tokens[i] = tokens[i].replace('"', ' ').strip()
193.             data = tokens[2].split("-")
194.
195.             medico = Medico.objects.get(cedula=tokens[0])
196.
197.
198.             utente = Utente.objects.get(numutente=tokens[1])
199.             fichamedica = Ficha_medica.objects.get(utente=utente)
200.
201.             consulta = Consulta(data=date(int(data[0]), int(data[1]), int(data[2])), medico
=medico,
202.                                     observacoes=tokens[3], fichamedica = fichamedica)
203.             consulta.save()
204.
205.             for i in range(4,len(tokens)):
206.                 token = tokens[i]
207.                 token = token.replace('[', ' ').replace(']', ' ').strip()
208.                 rtokens = token.split(",")
209.                 for j in range(len(rtokens)):
210.                     rtokens[j] = rtokens[j].replace('\\', ' ').strip()
211.
212.                 data1 = rtokens[0].split("-")
213.
214.
215.                 medicamento = Medicamento.objects.get(id_medicamento = rtokens[1])
216.
217.                 prescricao = Prescricao(data=date(int(data1[0]), int(data1[1]),
int(data1[2])), toma = rtokens[2])
218.                 prescricao.consulta = consulta
219.                 prescricao.save()
220.
221.                 prescricao.med.add(medicamento)
222.             k+=1
223.             string = "Consultas Povoadas!"
224.             return render(request, "gestor/erro.html", {'string': string, "username": username}
)

```

B. LOGINS

```

1. def login_medico(request):
2.     if request.method == "POST":
3.         form = AuthenticationForm(request, data=request.POST)
4.
5.         if form.is_valid():
6.             username = form.cleaned_data.get('username')
7.             password = form.cleaned_data.get('password')
8.             user = authenticate(username=username, password=password)
9.             if user is not None:
10.                 login(request, user)
11.                 messages.info(request, f"you are now logged in as {username} ")
12.                 return redirect('/alves_lda/medico/'+username)
13.             else:
14.                 messages.error(request, "Invalid username or password")
15.         else:
16.             messages.error(request, "Invalid username or password")
17.         form = AuthenticationForm()
18.         return render(request, 'medico/login_medico.html', {'form': form})
19.
20.
21. def registrar_medico(request):
22.     if request.method == "POST":
23.         form = NewUserForm(request.POST)
24.
25.         if form.is_valid():
26.             user = form.save()
27.             username = form.cleaned_data.get('username')
28.             x = username.replace("M", "")
29.             if "M" in username and Medico.objects.filter(cedula=x).exists():
30.                 messages.success(request, f"New Account Created: {username}")
31.                 login(request, user)
32.                 messages.info(request, f"You are now logged in as {username}")
33.                 return redirect("/alves_lda/")
34.             else:
35.                 for msg in form.error_messages:
36.                     messages.error(request, f"{msg}: {form.error_messages[msg]}")
37.         form = NewUserForm()
38.         return render(request, "medico/registrar_medico.html", context={"form": form})
39.
40.
41. def login_funcionario(request):
42.     if request.method == "POST":
43.         form = AuthenticationForm(request, data=request.POST)
44.
45.         if form.is_valid():
46.             username = form.cleaned_data.get('username')
47.             password = form.cleaned_data.get('password')
48.             user = authenticate(username=username, password=password)
49.             if user is not None:
50.                 login(request, user)
51.                 messages.info(request, f"you are now logged in as {username} ")
52.                 return redirect('/alves_lda/funcionario/'+username)
53.             else:
54.                 messages.error(request, "Invalid username or password")
55.         else:
56.             messages.error(request, "Invalid username or password")
57.         form = AuthenticationForm()
58.         return render(request, 'funcionario/login_funcionario.html', {'form': form})
59.
60.
61. def registrar_funcionario(request):

```

```

62.     if request.method == "POST":
63.         form = NewUserForm(request.POST)
64.
65.         if form.is_valid():
66.             user = form.save()
67.             username = form.cleaned_data.get('username')
68.             x = username.replace("F", "")
69.             if "F" in username and Funcionario.objects.filter(numfunc=int(x)).exists():
70.                 messages.success(request, f"New Account Created: {username}")
71.                 login(request, user)
72.                 messages.info(request, f"You are now logged in as {username}")
73.                 return redirect("/alves_lda")
74.         else:
75.             for msg in form.error_messages:
76.                 messages.error(request, f"{msg}: {form.error_messages[msg]}")
77.         form = NewUserForm()
78.         return render(request, "funcionario/registrar_funcionario.html",
79.             context={"form": form})
80.
81. def login_gestor(request):
82.     if request.method == "POST":
83.         form = AuthenticationForm(request, data=request.POST)
84.
85.         if form.is_valid():
86.             username = form.cleaned_data.get('username')
87.             password = form.cleaned_data.get('password')
88.             user = authenticate(username=username, password=password)
89.             if user is not None:
90.                 login(request, user)
91.                 messages.info(request, f"you are now logged in as {username} ")
92.                 return redirect('/alves_lda/gestor/'+username)
93.             else:
94.                 messages.error(request, "Invalid username or password")
95.         else:
96.             messages.error(request, "Invalid username or password")
97.         form = AuthenticationForm()
98.         return render(request, 'gestor/login_gestor.html', {'form': form})
99.
100. def registrar_gestor(request):
101.     if request.method == "POST":
102.         form = NewUserForm(request.POST)
103.
104.         if form.is_valid():
105.             user = form.save()
106.             username = form.cleaned_data.get('username')
107.             if "G" in username:
108.                 messages.success(request, f"New Account Created: {username}")
109.                 login(request, user)
110.                 messages.info(request, f"You are now logged in as {username}")
111.                 return redirect("/alves_lda")
112.             else:
113.                 for msg in form.error_messages:
114.                     messages.error(request, f"{msg}: {form.error_messages[msg]}")
115.             form = NewUserForm()
116.             return render(request, "gestor/registrar_gestor.html", context={"form": form})

```

C. FORMS

```

15. from django import forms
16. from django.contrib.auth.forms import UserCreationForm, AuthenticationForm
17. from django.contrib.auth.models import User
18. from .models import *
19. from django.forms import ModelForm
20.
21.
22. class RegistrarutenteModelForm(ModelForm):
23.     class Meta:
24.         model = Utente
25.         fields = ['nome', 'morada', 'nif', 'cc', 'datanasc', 'numutente', 'telefone',
26. 'telefone_emergencia', 'email']
27.
28. class HistoricoConsultasForm(forms.Form):
29.     numutente = forms.CharField(label = "Número de utente")
30.
31. class AgendaMedicoForm(forms.Form):
32.     data = forms.DateField(label = "Data")
33.
34.
35. class FichasmedicasForm(forms.Form):
36.     cedula = forms.CharField(label = "Cédula do médico")
37.
38.
39. class RealizarConsultaForm(forms.Form):
40.     numutente = forms.CharField(label = "Número de utente")
41.     data = forms.DateField(label="Data")
42.     hora = forms.TimeField(label = "Hora")
43.     cedula = forms.CharField(label="Cédula do médico")
44.
45.
46. class MarcarExameForm(forms.Form):
47.     tipo = forms.CharField(label = "Tipo")
48.     local = forms.CharField(label="Local")
49.     data_hora = forms.DateTimeField(label = "Data e Hora")
50.     duracao = forms.TimeField(label="Duração")
51.     preco = forms.DecimalField(label="Preço")
52.     estado = forms.BooleanField(label="Estado")
53.     observacoes = forms.CharField(label="Observações")
54.
55. class MedicoesForm(forms.Form):
56.
57.     data = forms.DateField(label = "Data")
58.     peso = forms.DecimalField(label = "Peso")
59.     glicemia = forms.DecimalField(label = "Glicemia")
60.     altura = forms.IntegerField(label = "Altura")
61.     tensaoarterial = forms.IntegerField(label = "Tensão arterial")
62.     colesterol = forms.IntegerField(label = "Colesterol")
63.     trigliceridios = forms.IntegerField(label = "Trigliceridios")
64.     saturacao = forms.IntegerField(label = "Saturação")
65.     inr = forms.IntegerField(label = "INR")
66.
67.
68. class PrescricoesForm(forms.Form):
69.
70.     data = forms.DateField(label = "Data")
71.     medicamento = forms.CharField(label="Medicamento")
72.     dosagem = forms.CharField(label="Dosagem")
73.     toma = forms.CharField(label = "Toma")
74.
75.
76.
77.

```



```

78. class ObservacoesForm(forms.Form):
79.     observacoes = forms.CharField(label = "Observações")
80.
81. class HistoricoForm(forms.Form):
82.     historico = forms.CharField(label = "Historico")
83.
84. class UtenteForm(forms.Form):
85.     nome = forms.CharField(label="Nome")
86.     morada = forms.CharField(label="Morada")
87.     nif = forms.CharField(label="NIF")
88.     cc = forms.CharField(label="CC")
89.     datanasc = forms.DateField(label="Data de nascimento")
90.     numutente = forms.CharField(label="Numero de utente")
91.     telefone = forms.CharField(label="Telefone")
92.     telefone_emergencia = forms.CharField(label="Telefone de emergência")
93.     email = forms.CharField(label="Email")
94.
95. class MedicamentoNomeForm(forms.Form):
96.     nome = forms.CharField(label="Nome")
97.
98. class FuncionarioForm(forms.Form):
99.     nome = forms.CharField(label="Nome do funcionário")
100.    morada = forms.CharField(label="Morada")
101.    nif = forms.CharField(label="NIF")
102.    cc = forms.CharField(label="CC")
103.    datanasc = forms.DateField(label="Data de nascimento")
104.    numfunc = forms.IntegerField(label="Número do funcionário")
105.
106. class MedicoForm(forms.Form):
107.    nome = forms.CharField(label="Nome do médico")
108.    morada = forms.CharField(label="Morada")
109.    nif = forms.CharField(label="NIF")
110.    cc = forms.CharField(label="CC")
111.    datanasc = forms.DateField(label="Data de nascimento")
112.    cedula = forms.CharField(label="Cédula")
113.    especialidade = forms.CharField(label="Especialidade")
114.
115. class MedicamentoForm(forms.Form):
116.    dci = forms.CharField(label="DCI")
117.    nome = forms.CharField(label="Nome")
118.    formafarmacaceutica = forms.CharField(label="Forma farmacêutica")
119.    dosagem = forms.CharField(label="Dosagem")
120.    estadoautorizacao = forms.CharField(label="Estado de autorização")
121.    generico = forms.BooleanField(label="Genérico")
122.    titular_aim = forms.CharField(label="Titular AIM")
123.
124. class UtenteIdadeForm(forms.Form):
125.    ano = forms.CharField(label="Ano de Nascimento")
126.
127. class ConsultaForm(forms.Form):
128.    id_con = forms.CharField(label="ID da Consulta")
129.
130. class PrescricaoForm(forms.Form):
131.    id_presc = forms.CharField(label="ID da Prescricao")
132.
133. class FicheiroForm(forms.Form):
134.    ficheiro = forms.CharField(label="Path do Ficheiro")

```