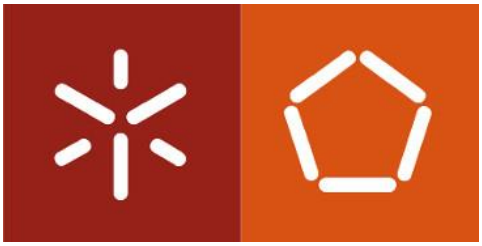
UNIVERSIDADE DO MINHO



Trabalho Prático II

Mestrado Integrado em Engenharia Biomédica

Aplicações Distribuídas

(1º Semestre/ Ano Letivo 2021/2022)

Realizado por:

Lara Alexandra Pereira Novo Martins Vaz (A88362)

Mariana Lindo Carvalho (A88360)

Tiago Miguel Parente Novais (A88397)

Braga

21 de janeiro de 2022

Resumo

Este trabalho tem como principal objetivo a familiarização com o *framework* de desenvolvimento de aplicações Django REST. Neste sentido, foi desenvolvido um sistema *multi-user* de gestão hospitalar onde, existe a possibilidade do registo de dados baseados em atos hospitalares, dos quais, atos médicos, atos farmacêuticos e atos de enfermagem. Durante o desenvolvimento deste trabalho, foram definidas entidades bem como as relações estabelecidas entre elas. Adicionalmente, procedeu-se à implementação das funcionalidades.

Índice

1. Introdução

Este trabalho tem como principal objetivo a familiarização com o *framework* de desenvolvimento de aplicações Django. Neste sentido, foi desenvolvido um sistema *multi-user* de gestão hospitalar onde existe a possibilidade do registo de dados baseados em atos hospitalares, dos quais, atos médicos (registo de consultas), atos farmacêuticos e atos de enfermagem. De forma complementar, o sistema permite também a gestão do stock de medicamentos e de outros artigos.

Sendo este um tema de interesse na área da Engenharia Biomédica, foi solicitada a implementação de dois sistemas, um responsável pela gestão dos atos médicos, farmacêuticos e de enfermagem e outro que engloba toda a informação acerca do stock de medicamentos e outros artigos diversos. Primeiramente, procedeu-se ao levantamento das condições necessárias para o desenvolvimento das aplicações envolvidas no processo de gestão. Assim, a etapa inicial centrou-se na definição das entidades envolvidas no projeto, bem como das interfaces dos diferentes serviços idealizados, considerando a lógica da gestão de negócio necessária para um correto funcionamento das aplicações.

Para a construção deste caso prático, é necessário cumprir os seguintes requisitos:

- Levantamento dos requisitos e definição das funcionalidades a desenvolver;
- Definição das entidades envolvidas;
- Definição das interfaces dos diferentes serviços que idealizar;
- Implementação da lógica de negócio necessárias ao correto funcionamento do sistema;
- Implementação do *backend* e *frontend* Web de suporte à aplicação.

2. Preliminares

2.1. Django

O Django é um *framework* baseado na linguagem *Python* e, corresponde a uma abstração que unifica códigos comuns entre vários projetos de software, disponibilizando uma funcionalidade genérica. Assim, permite ao utilizador criar aplicações Web, que consistem num conjunto de elementos que permitem o desenvolvimento de sites de forma mais rápida e fácil.

Cada projeto é acompanhado por um conjunto de diretorias e ficheiros pré-definidos, tais como:

• *settings*: contém todas as configurações da instalação do Django, nas quais são especificadas as aplicações instaladas e a base de dados a utilizar [1];

• *models*: corresponde à fonte de toda a informação. Cada modelo é mapeado para uma única tabela de base de dados [2];

• *views:* correspondem a funções que armazenam toda a lógica responsável pelo processamento do pedido de um utilizador, de forma a retornar a resposta pretendida [3];

• *forms*: permitem que os Websites recebam input dos utilizadores, ou seja, que o utilizador possa inserir texto, selecionar opções e manipular objetos [4];

• *templates*: contém as partes estáticas da saída HTML desejadas e a sintaxe que descrever como o conteúdo dinâmico será inserido [5];

• *URLs*: quando um utilizador faz um pedido de uma página, o servidor procura a respetiva *views* através dos *URL*s e depois retorna a devida resposta, tendo em conta se encontrou ou não o que foi pedido [6].

2.2. Django REST framework

O Django REST *framework* é um kit de ferramentas poderoso e flexível para construir APIs da Web [7].

Alguns motivos pelos quais se utiliza esta *framework* são:

- A API navegável na Web é uma grande vitória de usabilidade para seus desenvolvedores;
- Políticas de autenticação, incluindo pacotes para OAuth1a e OAuth2 ;

- Serialização que dá suporte a fontes de dados ORM e non-ORM;
- Totalmente personalizável - basta usar visualizações regulares baseadas em funções, caso não sejam necessários recursos mais poderosos;
- Extensa documentação e excelente suporte da comunidade;
- Confiável e usado por empresas reconhecidas internacionalmente, incluindo Mozilla , Red Hat , Heroku e Eventbrite [7].

2.3. Arquitetura Cliente-Servidor

A arquitetura cliente-servidor é um tipo de estrutura de aplicações distribuídas que divide tarefas entre um serviço (servidor) e os que pretendem requisitar o serviço (clientes).

O Django é um sistema de pedidos (*requests*) e respostas (*responses*), em que o processo pode ser descrito segundo a seguinte sequência de passos:

1. O *urls.py* faz o mapeamento do URL pedido para uma das *views* e chama-a. No caso do *caching* estar ativo, essa *view* pode verificar se a versão da página existente guardada em cache existe, avança todos os passos seguintes e retorna essa versão;
2. No caso de não se verificar a situação acima descrita, a *view* executa a ação de pedido (*request*);
3. O *model* define a informação em *Python* e interage com ela;
4. Após a execução da tarefa pedida, a *view* retorna um objeto de resposta HTTP para o Web browser;
5. Um *template* geralmente retorna páginas HTML.

3. Desenvolvimento do Sistema de Gestão

De forma a utilizar as funcionalidades do Django, foi necessário criar um projeto que abordasse todo o *setup* inicial e funcionasse como “esqueleto” do trabalho. Para tal, foi auto gerada uma coleção de definições no Django onde estão incluídas:

- Configuração da base de dados;
- Opções específicas do Django;
- Definições das aplicações.

Neste trabalho prático recorreu-se à base de dados SQLite3.

Uma vez criada a aplicação, de seguida, foram elaborados os *models*, *views*, *forms*, *templates* e *URLs* que são responsáveis por todo o funcionamento e apresentação da aplicação.

O sistema elaborado é constituído por onze entidades: **Utilizador** (equivalente ao User do Django), **Utente**, **Medico**, **Enfermeiro**, **Farmaceutico**, **Funcionario,** **Medicamento**, **OutrosArtigos**, **AtoMedico**, **AtoFarmaceutico** e **AtoEnfermagem**.

O Sistema de Gestão do Processo Clínico (**GCP)** tem como utilizadores os **Medicos**[1], os **Enfermeiros**, os **Utentes** e o **Funcionario** e contém informações relativas aos **AtosMedicos** e **AtosEnfermagem** e aos respetivos participantes desses atos: **Utentes, Medicos, Enfermeiros**. Neste sistema estão registados todos os atos médicos que incluem dispensa de medicamentos e de outros artigos (como analgésicos e antibióticos, entre outros) a um utente. Já os atos de enfermagem apenas incluem dispensa de outros artigos a um utente, uma vez que um enfermeiro não pode fazer dispensa de medicamentos. Cada ato regista a identificação do utente, a identificação do agente (médico ou enfermeiro), uma data/hora, um artigo e/ou medicamento e respetiva quantidade utilizada.

Relativamente às funcionalidades a que os utilizadores têm acesso no GPC, os funcionários **Medico** e **Enfermeiro** podem criar atos médicos e de enfermagem, respetivamente, consultar os seus dados e quais os atos que já prestaram. Por sua vez, os **Utentes** apenas podem aceder aos seus próprios dados e visualizar os atos que lhes foram prestados. Relativamente ao **Funcionario,** estes é um utilizador especial que faz parte do staff, tendo assim os privilégios de um administrador, mas podendo também executar pedidos no lado do *user*. Neste lado do *user*, o **Funcionario** pode visualizar estatísticas relevantes relativas aos atos e seus utilizadores. Este sistema de gestão é implementado de forma que:

• a um **AtoMedico/AtoEnfermagem** corresponde um e um só utente;

• a um **Medico** e a um **Utente** pode corresponder mais do que um **AtoMedico/AtoEnfermagem**, isto é, o mesmo médico ou enfermeiro pode efetuar mais do que um ato.

Quanto ao Sistema de Gestão Logística e de Medicamentos (**GLM**), os seus utilizadores são os **Farmaceuticos**, sendo que este sistema contém informações relativas

[1] Apesar de não existir nenhum modelo Medicos, como a palavra no plural visa representar vários objetos do modelo Medico, então, nomes de modelos no plural também estão assinalados a negrito.

aos participantes (**Farmaceuticos**), aos seus atos e também informações relativas aos stocks dos **Medicamentos** e **OutrosArtigos**. Neste sistema é também possível registar aquisições, adicionar novos **Medicamentos** e **OutrosArtigos** e verificar quais os **Medicamentos** e **OutrosArtigos** que estão com stock baixo.

3.1. *Models*

3.1.1. Utilizador

O modelo **Utilizador** corresponde aos atributos comuns que todos os *users* possuem. Estes atributos dizem respeito ao *nome*, *BI*, *NIF*, *morada* e *codigo_postal*. Pretende-se que o NIF e o BI sejam únicos na base de dados e desta forma, esses atributos têm a opção *unique* = True. Neste modelo, assim como em todos existe também o seu id, que é definido automaticamente pelo Django.

3.1.2 Medico

Para definir a entidade **Medico** utilizaram-se os mesmo campos do **Utilizador** com a adição dos campos *cedula* e *especialidade*. Definiu-se neste modelo e nos seguintes, a função __str__, que permite visualizar o conteúdo de cada objeto guardado.

3.1.3 Enfermeiro

Para definir a entidade **Enfermeiro** utilizaram-se os mesmos campos do **Utilizador** com a adição do campo especialidade.

3.1.4 Farmacêutico

Para definir a entidade **Farmacêutico** utilizaram-se o mesmo campo do **Utilizador**.

3.1.5 Utente

Para definir a entidade **Utente** utilizaram-se o mesmo campo do **Utilizador**.

3.1.6 Funcionario

Para definir a entidade **Funcionario** utilizaram-se o mesmo campo do **Utilizador**.

3.1.7 Medicamento

Para o modelo **Medicamento** utilizaram-se os atributos *dci, nome_medicamento, forma_farmaceutica, dosagem, estado_autorizacao, generico* e *titular_aim.*

3.1.8 Outro_Artigo

Para o modelo **Outro_Artigo** utilizaram-se os atributos *nome_artigo* e *fornecedor*.

3.1.9 Stock_med

Para o modelo **Stock_med** utilizaram-se os atributos *id_med* (este atributo é *ForeignKey* uma vez que se pretende associar o stock aos objetos **Medicamento**) e *quant* que corresponde à respetiva quantidade desse **Medicamento**.

3.1.10 Stock_art

Para o modelo **Stock_art** utilizaram-se os atributos *id_art* (este atributo é *ForeignKey* uma vez que se pretende associar o stock aos objetos **Outro_Artigo**) e *quant* que corresponde à respetiva quantidade desse artigo.

3.1.11 Ato_Medico

Para o modelo **Ato_Medico** utilizaram-se os atributos *medico* (este atributo é *ForeignKey* uma vez que um **Medico** pode ter vários atos associados – relação many-to-one), *utente* (este atributo é *ForeignKey* uma vez que um **Utente** pode ter vários atos associados – relação many-to-one), a *hora* que corresponde à data/hora de quando o ato foi efetuado, a *quant_med* que é o número de unidades de medicamento que foi utilizado, o *medicamento* (este atributo é *ForeignKey* uma vez que se pretende associar o medicamento utilizado ao respetivo objeto **Medicamento**), a *quant_art* que é o número

de unidades do artigo que foi utilizado e o *outroartigo* (este atributo é *ForeignKey* uma vez que se pretende associar o artigo utilizado ao respetivo objeto **Outro_Artigo**).

3.1.12 Ato_Enfermagem

Para o modelo **Ato_Enfermagem** utilizaram-se os atributos *enfermeiro* (este atributo é *ForeignKey* uma vez que um **Enfermeiro** pode ter vários atos associados – relação many-to-one), *utente* (este atributo é *ForeignKey* uma vez que um **Utente** pode ter vários atos associados – relação many-to-one), a *hora* que corresponde à data/hora de quando o ato foi efetuado, a *quant_art* que é o número de unidades do artigo que foi utilizado e o *outroartigo* (este atributo é *ForeignKey* uma vez que se pretende associar o artigo utilizado ao respetivo objeto **Outro_Artigo**).

3.1.13 Ato_Medico

Para o modelo **Ato_Farmaceutico** utilizaram-se os atributos *medico* (este atributo é *ForeignKey* uma vez que um **Farmaceutico** pode ter vários atos associados – relação many-to-one), a *hora* que corresponde à data/hora de quando o ato foi efetuado, a *quant-_med* que é o número de unidades de medicamento que foi utilizado, o *medicamento* (este atributo é *ForeignKey* uma vez que se pretende associar o medicamento utilizado ao respetivo objeto **Medicamento**), a *quant_art* que é o número de unidades do artigo que foi utilizado e o *outroartigo* (este atributo é *ForeignKey* uma vez que se pretende associar o artigo utilizado ao respetivo objeto **Outro_Artigo**).

3.2 *Serializers*

No Django REST *framework*, os *serializers* são responsáveis por converter objetos em tipos de dados compreensíveis por javascript e frameworks front-end. Os *serializers* também fornecem desserialização, permitindo que os dados analisados sejam convertidos novamente em tipos complexos, depois de validar primeiro os dados recebidos. Os *serializers* no *framework* REST funcionam de forma muito semelhante às classes Form e ModelForm do Django. Neste trabalho foram utilizados os serializers ModelSerializer [8].

As classes *serializers* desenvolvidas fornecem um atalho que permitem criar automaticamente uma classe *Serializer* com campos que correspondem aos campos do respetivo *Model*. A classe *ModelSerializer* é a mesma que uma classe *Serializer* normal, exceto que gera automaticamente um conjunto de campos com base no modelo.

3.3 *Views, Forms* e *Templates*

Uma *view* é uma função que recebe um pedido Web e retorna uma resposta Web, uma vez que contém a lógica arbitrária necessária para retornar a resposta ao pedido. Essa resposta pode ser o conteúdo HTML de uma página Web, um redireccionamento ou até mesmo um erro 404.

Na maioria das *views* foram implementados *forms* para possibilitar a interação dos **Utilizadores** com a aplicação, podendo, assim, fazer pedidos consoante o que pretendem requisitar.

Todas as *views* exceto as de login e logout requerem que o utilizador esteja autenticado. Os *templates*, por sua vez, apenas estão visíveis para grupos específicos, por exemplo, o menuutente.html apenas é visível para utilizadores que pertençam ao grupo "Utentes".

3.3.1 *ViewSets*

O Django REST *framework* permite combinar a lógica de um conjunto de *views* relacionadas numa única classe, designada de *ViewSet*. A *ViewSet* é simplesmente um tipo de *View* baseada em classe, que não fornece nenhum manipulador de método como .get() ou .post(), em vez disso, fornece ações como .list() e .create().

Neste trabalho foram desenvolvidos vários *ModelViewSet* cujo objetivo é visualizar e editar a entidade em questão, como o *User*, o *Medicamento*, *Outro_Artigo*, entre outros.

3.3.2 *Login*

Para o login é utilizado o *form LoginForm*, onde é requerido um *username* e *password*. No caso de algum desses campos não estar previamente guardado ou não corresponder ao guardado na base de dados, é apresentada a mensagem "Incorrect username or password! Please, try again.".

Caso o *form* seja válido, isto é, se tanto o *username* e a *password* estiverem autenticados, verifica-se a presença de um utilizador autenticado. Assim, dependendo a que grupo pertence o utilizador, este é redirecionado para o seu menu. Caso, o utilizador seja um Utente, por exemplo, o mesmo é redirecionado para o "Menu Utente".

3.3.3 *Logout*

A view do *logout* utilizada corresponde à predefinida pelo Django. Quando se faz *logout*, toda a informação da sessão é removida para prevenir que outra pessoa, ao utilizar o mesmo web browser para fazer login, não tenha acesso à informação da sessão anterior. De seguida, é-se redirecionado para a página do login, com uma mensagem "Logout efetuado.".

3.3.4 *AddGroup*

A *view* do *AddGroup* utilizada corresponde à predefinida pelo Django. Quando se faz *logout*, toda a informação da sessão é removida para prevenir que outra pessoa, ao utilizar o mesmo web browser para fazer login, não tenha acesso à informação da sessão anterior. De seguida, é-se redirecionado para a página do login, com uma mensagem "Logout efetuado.".

3.3.5 *Views* dos menus

As *views* menu(utilizador) permitem que os utilizadores acedam ao seu menu. O template *menuutente.html* tem ligações para visualizar a sua ficha de utente. O template *menumedico.html* tem ligações para visualizar as suas informações, os atos prestados e permite também criar atos médicos. O template *menuenfermeiro.html* tem ligações para visualizar as suas informações, os atos prestados e permite também criar atos de enfermagem. O template *menufarmaceutico.html* tem ligações para visualizar as suas

informações, os atos prestados e os stocks, permite também criar atos farmacêuticos, adicionar novos medicamentos e artigos e ver quais os medicamentos e artigos que estão com baixo stock. O template *menufuncionario.html* tem ligações para visualizar estatísticas relevantes.

3.3.6 *View* relativa ao Utente

Para que um utente aceda à sua ficha, foi criada a *view ver_ficha_utente*. É obtido o *id* do utente logado que realizou o pedido, e através do id obtém-se objeto **Utente** respetivo. Depois, obtêm-se os atos médicos e de enfermagem desse Utente. Finalmente, o pedido é renderizado no template fichautente.html com o *context* (dicionário que contém todas as informações obtidas).

3.3.7 *Views* relativas aos stocks

O objetivo destas *views* é fornecer um conjunto de métodos auxiliares que permitem a redução ou aumento de stock de **Medicamentos** e de **OutrosArtigos** automaticamente, aquando da realização de atos.

3.3.8 *Views* relativas ao Funcionário

Nestas *views* existe um conjunto de métodos cujo objetivo é apresentar estatísticas relativas aos **Medicos, Enfermeiros** e **Farmacêuticos** e respetivos atos e aos **Utentes.** Tomando como exemplo a *view farmacêutico_estatisticas*, nesta são obtidos o total de **Atos_Farmaceuticos** realizados e o total de **Farmaceuticos** e calculou-se o número médio de atos realizados por **Farmaceutico**. De seguida, através de um ciclo *for* determinou-se qual o **Farmaceutico** que realizou mais atos. Posteriormente, determinou-se o Top 3 de **Medicamentos** e de **OutrosArtigos** com mais stock. Finalmente, o pedido é renderizado no template farmaceuticoestatisticas.html com o *context* (dicionário que contém todas as informações obtidas).

3.3.9 *Views* relativas ao Médico

3.3.9.1 Criar Atos Médicos

A *view ato_medico_create* utiliza o form *FormAtoMedico*, que leva como argumentos um **Medico,** um **Utente**, a hora a que ocorreu o ato, a quantidade de **Medicamentos** utilizados (ou receitados), o respetivo **Medicamento**, a quantidade de **OutrosArtigos** utilizados e o respetivo artigo. É ainda de realçar que apesar do indivíduo que preenche o formulário não colocar lá nenhum id, todos os formulários preenchidos levam à criação de objetos que têm sempre um id associado e criado automaticamente pelo Django.

Se o *form* for válido e se não existir já nenhum ato criado nessa hora, então o form é guardado, criando-se um novo **Ato_Medico.** São ainda obtidos os ids dos **Medicamento e OutroArtigo** usados, e respetivas quantidades, de forma a passar esses argumentos aos métodos auxiliares de redução instantânea de stock. Por último, é feito o *return* do *sucesso.html* que contém uma mensagem de sucesso no registo do ato.

Caso já exista um ato criado àquela hora, então verifica-se se esse ato contém o mesmo **Medico** e **Utente**. Se não tiver, então guarda-se o *form* e repetem-se os mesmos passos mencionados no parágrafo anterior. Se nenhuma dessas condições se verificar, então, é retornado o *erro.html* que contém uma mensagem de erro.

Finalmente, o pedido é renderizado no template *atomedico_create.html*.

3.3.9.2 Ver a informação do Médico

A *view info_medico* obtém o id do **Medico** que fez o pedido e apresenta as suas informações que são adicionadas no dicionário *context* e renderizadas no *infomedico.html.*

3.3.9.3 Consultar os Atos Médicos

A *view medico_atos* obtém o id do **Medico** que fez o pedido e apresenta os atos que este já prestou. Estes são adicionados ao dicionário *context* e renderizados no *medicoatos.html.*

3.3.10 *Views* relativas ao Enfermeiro

3.3.10.1 Criar Atos de Enfermagem

A *view atoenfermagem_create* utiliza o form *FormAtoEnfermagem*, que leva como argumentos um **Enfermeiro,** um **Utente**, a hora a que ocorreu o ato, a quantidade de **OutrosArtigos** utilizados e o respetivo artigo.

Se o *form* for válido e se não existir já nenhum ato criado nessa hora, então o form é guardado, criando-se um novo **Ato_Enfermagem.** São ainda obtidos o id do **OutroArtigo** usado e respetiva quantidade, de forma a passar esses argumentos aos métodos auxiliares de redução instantânea de stock. Por último, é feito o *return* do *sucesso.html* que contém uma mensagem de sucesso no registo do ato.

Caso já exista um ato criado àquela hora, então verifica-se se esse ato contém o mesmo **Enfermeiro** e **Utente**. Se não tiver, então guarda-se o *form* e repetem-se os mesmos passos mencionados no parágrafo anterior. Se nenhuma dessas condições se verificar, então, é retornado o *erro.html* que contém uma mensagem de erro.

Finalmente, o pedido é renderizado no template *atoenfermagem_create.html*.

3.3.10.2 Ver a informação do Enfermeiro

A *view info_medico* obtém o id do **Enfermeiro** que fez o pedido e apresenta as suas informações que são adicionadas no dicionário *context* e renderizadas no *enfermerio_info.html.*

3.3.10.3 Consultar os Atos de Enfermagem

A *view enfermeiro_atos* obtém o id do **Enfermeiro** que fez o pedido e apresenta os atos que este já prestou. Estes são adicionados ao dicionário *context* e renderizados no *enfermeiroatos.html.*

3.3.11 *Views* relativas ao Farmacêutico

3.3.11.1 Criar Atos Farmacêuticos

A *view atofarmaceutico_create* utiliza o form *FormAtoFarmaceutico*, que leva como argumentos um **Farmacêutico,** a hora a que ocorreu o ato, a quantidade de

Medicamentos a repor em stock, o respetivo **Medicamento**, a quantidade de **OutrosArtigos** a repor em stock e o respetivo artigo.

Se o *form* for válido e se não existir já nenhum ato criado nessa hora, então o form é guardado, criando-se um novo **Ato_Farmaceutico.** São ainda obtidos os ids dos **Medicamento e OutroArtigo** usados, e respetivas quantidades, de forma a passar esses argumentos aos métodos auxiliares de aumento instantâneo de stock. Por último, é feito o *return* do *sucesso.html* que contém uma mensagem de sucesso no registo do ato.

Caso já exista um ato criado àquela hora, então verifica-se se esse ato contém o mesmo **Farmacêutico**. Se não tiver, então guarda-se o *form* e repetem-se os mesmos passos mencionados no parágrafo anterior. Se nenhuma dessas condições se verificar, então, é retornado o *erro.html* que contém uma mensagem de erro.

Finalmente, o pedido é renderizado no template *atofarmaceutico_create.html*.

3.3.11.2 Ver a informação do Farmacêutico

A *view info_farmaceutico* obtém o id do **Farmaceutico** que fez o pedido e apresenta as suas informações que são adicionadas no dicionário *context* e renderizadas no *infofarmaceutico.html.*

3.3.11.3 Consultar os Atos Médicos

A *view farmaceutico_atos* obtém o id do **Farmaceutico** que fez o pedido e apresenta os atos que este já prestou. Estes são adicionados ao dicionário *context* e renderizados no *farmaceuticoatos.html.*

3.3.11.4 Alerta de baixos stocks

A *view farmaceutico_alerta* obtém os **Medicamentos** em stock cuja quantidade é inferior a 250 unidades e apresenta as informações do respetivo **Medicamento** e quantidade associada no dicionário *context.* O mesmo raciocínio é aplicado para os **OutrosArtigos.** Por último, as informações que estão no *context* são renderizadas no *farmaceuticoalerta.html.*

3.3.11.5 Adicionar novos Medicamentos

A *view adiciona_medicamento* utiliza o form *FormMedicamento*, que leva como argumentos o nome comum do **Medicamento**, o seu nome científico, a sua forma farmacêutica, a sua dosagem, o seu estado de autorização, um parâmetro que indica se o **Medicamento** é ou não genérico e ainda o seu fornecedor.

Se o *form* for válido e se não existir já nenhum **Medicamento** igual na base de dados, então o *form* é guardado, criando-se um novo **Medicamento.** De seguida, o **Medicamento** criado é adicionado ao **Stock_med** com um stock inicial de 505 unidades. Por último, é feito o *return* do *sucesso.html* que contém uma mensagem de sucesso no registo do ato.

Caso já exista um **Medicamento** igual ao que se pretende criar na base de dados, então é retornado o *erro.html* que contém uma mensagem de erro.

Finalmente, o pedido é renderizado no template *adiciona_medicamento.html*.

3.3.11.6 Adicionar novos Artigos

A *view adiciona_outro_artigo* utiliza o form *FormOutroArtigo*que leva como argumentos o nome do **OutroArtigo** e o seu fornecedor.

Se o *form* for válido e se não existir já nenhum **OutroArtigo** igual na base de dados, então o *form* é guardado, criando-se um novo **OutroArtigo.** De seguida, o **OutroArtigo** criado é adicionado ao **Stock_art** com um stock inicial de 505 unidades. Por último, é feito o *return* do *sucesso.html* que contém uma mensagem de sucesso no registo do ato.

Caso já exista um **OutroArtigo** igual ao que se pretende criar na base de dados, então é retornado o *erro.html* que contém uma mensagem de erro.

Finalmente, o pedido é renderizado no template *adiciona_outro_artigo.html*.

3.3.11.7 Consultar o Stock de Medicamentos

A *view farmaceutico_stock_med* obtém todos os **Medicamentos** existentes no stock e a sua quantidade e adiciona essa informação ao dicionário *context* que depois é renderizado no *farmaceuticostockmed.html*.

3.3.11.8 Consultar o Stock de Artigos

A *view farmaceutico_stock_art* obtém todos os **OutrosArtigos** existentes no stock e a sua quantidade e adiciona essa informação ao dicionário *context* que depois é renderizado no *farmaceuticostockart.html*.

3.4 URLs

Para a criação de determinada app no Django, é utilizado um módulo chamado URL que consiste num mapeamento entre as expressões URL (os caminhos) e as *views*. Neste ficheiro, estão indicados os URLs da aplicação hospital que foi criada, incluindo os caminhos do *login, logout* e de todos os menus.

3.5 *Admin*

Uma das partes mais poderosas do Django é a interface de administração automática. Este lê metadados dos seus modelos para fornecer uma interface rápida e centrada no modelo, onde superutilizadores podem gerir o conteúdo do seu site.

A interface de administração tem a capacidade de editar modelos na mesma página que um modelo "pai".

Para além disso, uma vez que foi personalizado o modelo de User e pretendia-se que este também funcionasse com o administrador, foi necessário registá-lo no admin.py. Assim, criaram-se mais 10 *forms (MedicoCreationForm, MedicoChangeForm, UtenteCreationForm, UtenteChangeForm, FuncionarioCreationForm e FuncionarioChangeForm, EnfermeiroCreationForm, EnfermeiroChangeForm, FarmaceuticoCreationForm, FarmaceuticoChangeForm,),* e recorreu-se ao BaseUserAdmin para criar o *MedicoAdmin, FuncionarioAdmin, UtenteAdmin EnfermeiroAdmin* e *FarmaceuticoAdmin.*

3.6 *Loads*

Foi criada uma diretoria denominada *loads* na qual estão presentes os ficheiros responsáveis pelo povoamento da base de dados relativamente aos modelos **Utente,**

Medico, Enfermeiro, Farmaceutico, Funcionário, Medicamento, OutroArtigo, Stock_art e **Stock_med**.

Na diretoria *loads* está também presente, o ficheiro *permissions_groups.py*, que permitiu a criação de cinco grupos de utilizadores - Médicos, Funcionário, Utentes, Enfermeiros e Farmacêuticos - e a atribuição das respetivas permissões a cada um desses grupos.

Assim, é de salientar que no povoamento dos **Utentes, Medicos, Enfermeiros, Farmaceuticos** e **Funcionario**, estes são adicionados ao respetivo grupo já criado. O *username* é igual ao nome do grupo em minúscula seguido do número do seu id.

4. Conclusão

O desenvolvimento deste trabalho prático permitiu consolidar os diversos conhecimentos relativos à linguagem *Python*, mais precisamente à *framework* Django REST.

A partir da utilização da base de dados, de uma arquitetura cliente-servidor e de todas as funcionalidades do *framework* em questão, foi possível criar um sistema de gestão do processo clínico e um sistema de gestão de logística e medicamentos, através da implementação de um sistema de registo e consulta de atos, de utilizadores e suas informações, de stocks e de estatísticas relevantes.

Ao longo da construção da aplicação, foram sentidas dificuldades na perceção do papel e relação entre cada uma das partes (*views, urls, templates*). Contudo, à medida que se foram criando *views*, passando *contex*t aos templates e efetuando pedidos através da colocação de URLs no browser a resolução do problema tornou-se mais clara.

Apesar de se ter conseguido atingir as metas essenciais para o trabalho, reconhecem-se aspetos a melhorar. Para além das funções atuais, poder-se-ia ter introduzido a possibilidade de alterar informações de um objeto.

Em suma, a aplicação desenvolvida permite o login dos utentes, médicos, enfermeiros, farmacêuticos e funcionários, tendo sido desenvolvido para cada um deles funcionalidades especificas e ainda a consulta, aquisição ou utilização de medicamentos e artigos presentes no stock.

Referências

[1] Django settings. https://docs.djangoproject.com/en/3.1/topics/settings/. Acedido a 19 jan 2022.

[2] Models. https://docs.djangoproject.com/en/3.1/topics/db/models/ Acedido a 19 jan 2022.

[3] Writing Views. https://docs.djangoproject.com/en/3.1/topics/http/views/ Acedido a 19 jan 2022.

[4] Working with forms. https://docs.djangoproject.com/en/3.1/topics/forms/ Acedido a 19 jan 2022.

[5] Templates. https://docs.djangoproject.com/en/3.1/topics/templates/ Acedido a 19 jan 2022.

[6] URL dispatcher. https://docs.djangoproject.com/en/3.1/topics/http/urls/ Acedido a 19 jan 2022.

[7] Django REST framework. https://www.django-rest-framework.org/. Acedido a 19 jan 2021.

[8] Geek for Geeks. https://www.geeksforgeeks.org/serializers-django-rest-framework/. Acedido a 19 jan 2021.