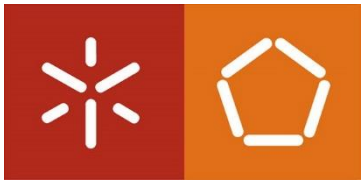

Mestrado em Engenharia Biomédica – Informática Médica

1º Ano | 2022/2023 | 1º Semestre

Aplicações Distribuídas

Trabalho Prático 2 - Sistema CODU

Grupo 4

Hugo Silva PG50416

Docentes: Gonçalo Carvalho PG50392

António Sousa Tomás Lima PG50788

Índice

Introdução

Tal como percebido no trabalho anterior, a criação de um serviço de atendimento de chamadas Centro de Orientação de Doentes Urgentes (CODU). Mesmo que bastante simplista, tem diversos passos a cumprir e objetivos a considerar. No entanto, na aplicação anterior seria necessário aceder ao terminal para ter a capacidade de usar a aplicação, assim como ter um conhecimento geral dos comandos a utilizar, entre outros problemas. Isto claramente não é ideal tendo em conta o utilizador comum, tornando assim o sistema inviável para aplicações de grande escala.

Devido a isto, a criação de uma aplicação que se possa considerar mais polida, ou seja, de uso para qualquer utilizador é necessária. Com este intuito, recorreu-se ao Django e à *framework* REST para resolver o problema.

Assim sendo, foi necessária a divisão da aplicação em *Frontend* e *Backend*. A implementação de ambas será explicada mais à frente, mas de uma forma geral, o *Backend* é a infraestrutura da aplicação, sendo composto pelo código da aplicação em si, do *server* e da base de dados. Por sua vez, o *Frontend* é a interface com a qual o utilizador vai interagir, tratando da componente gráfica e lógica da aplicação.

Uma grande porção do tinha já sido delineada anteriormente, mais precisamente, a arquitetura e as fundações que serão agora necessárias. Tendo isto em conta, diversas porções do trabalho anterior foram reaproveitadas, tal como pressupostos, arquitetura (com algumas modificações) e o funcionamento geral da aplicação, enquanto que a grande mudança foi efetivamente a criação de uma interface através da qual se comunica com o servidor e com a base de dados em vez de o fazer diretamente. Foi também necessário criar os mecanismos de gestão das bases de dados criadas pelo Django.

Planeamento e Estruturação

Quanto ao planeamento da base de dados, foi aproveitado o planeamento do Trabalho Prático 1, apenas sendo necessário adaptá-lo à biblioteca Django do Python, pelo que os pressupostos, regras e convenções propostas se mantêm. A título de exemplo, alguns desses pressupostos são:

- Cada evento apresenta apenas 1 vítima.
- O meio de socorro responsável pelo transporte retorna sempre a vítima para o seu próprio posto.
- O helicóptero consegue pousar em qualquer local para responder a eventos.

Como se verifica na figura 1, as relações entre as entidades foram alteradas para melhor espelhar a necessidade e funcionalidade de uma base de dados, principalmente com o intuito de tentar reduzir ao máximo a redundância de informação presente no sistema. As alterações efetuadas foram:

- Remoção da relação N:N entre as entidades Grau e Meio de Socorro, pois estes relacionam-se através da entidade Evento.
- Deixar explícito duas relações existentes entre o Meio de Socorro e o Evento, uma N:N que indica que um Meio pode responder a vários Eventos e que um Evento pode necessitar de vários Meios, mas também uma relação N:1, que indica que um meio apenas tem um Evento atual (ou seja, só pode responder a um evento de cada vez), mas que esse Evento atual pode necessitar na mesma de vários Meios.
- Remoção da superclasse Local, que consistia num sistema bidimensional de coordenadas x, y, que existia no Evento (com relação 1:N) e no Posto (sua subclasse), tendo sido optado por criar duas variáveis com essa informação em cada uma dessas entidades.

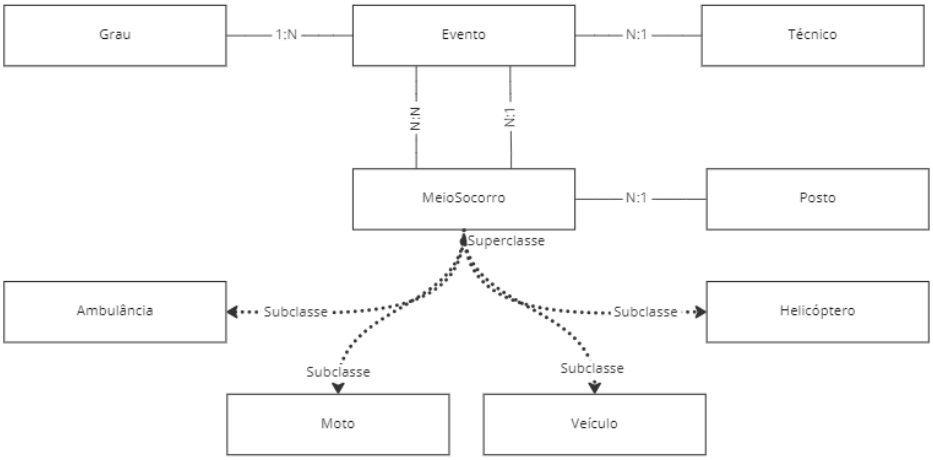

Figura 1 - Modelo de entidades e relações na fase de planeamento

Base de dados

De seguida, foi necessário alterar o tipo das variáveis, em Java, para o respetivo *Model Field*, do Django, ou seja: *String* para *CharField*, *int* para *IntegerField*, *boolean* para *BooleanField*, *LocalDate* para *DateField e LocalDateTime* para *DateTimeField*. Os atributos que continham os identificadores de outra classe foram alterados para *ForeignKey*, e os do tipo *ArrayList* e *HashMap* foram divididos em vários atributos, apagados, ou alterados para *ManyToManyField*, no caso de serem resultantes de relações N:N.

Por exemplo, o *HashMap totalMeiosEnviados* do Técnico, que tinha como *key* uma *String* do tipo do meio e *value* o número de meios desse tipo que ele mobilizou, foi dividido em 4 atributos do tipo *IntegerField*, que têm a mesma função. O *atendimentosEfetuados*, também do Técnico, que era um *ArrayList* com os Eventos que este atendeu, foi eliminado, pois como a relação Evento-Técnico é N:1, essa informação já está guardada como *ForeignKey* no Evento.

Quanto ao Meio de Socorro, o *ArrayList eventosAtendidos* foi alterado para um *ManyToManyField* e o *evento_atual* para uma *ForeignKey*, mas em vez de ser na superclasse, teve de ser colocado em cada subclasse, para, através da opção *related_name*, evitar conflitos ao guardar as *ForeignKey* de uma só entidade mais do que uma vez numa mesma entidade.

No final, a base de dados ficou como demonstrado na figura 2, na qual não são demonstrados os modelos da Moto, Veículo e Helicóptero para facilitar a sua leitura, pois têm os mesmos atributos e relações do modelo Ambulância.

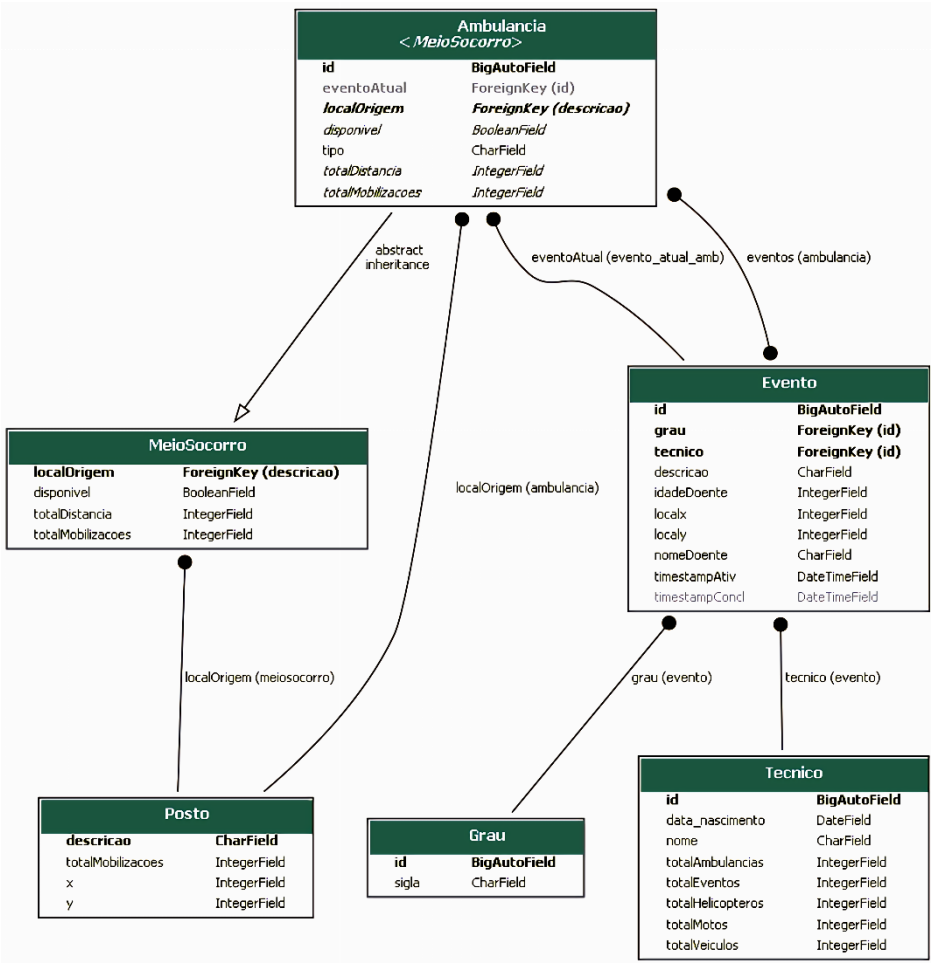

Figura 2 - Modelo de entidades e relações da base de dados

Também é de referir que o modelo Grau tem, no momento da criação da base de dados, todos os 4 valores possíveis (1 – SBV; 2 – SBV_U; 3 – SAV; 4 – SAV_U) atribuídos. Isto é conseguido através da inserção destas instâncias automaticamente por *hardcoding* na migração inicial da aplicação.

Backend

O *Backend* é responsável por fazer a comunicação entre o *Frontend* e as bases de dados, quase como um intermediário. Serve assim como uma barreira de segurança, não permitindo que ações dos utilizadores no Frontend afetem de imediato as bases de dados, comprometendo a sua integridade. Para realizar esta tarefa, recorreu-se ao uso de *serializers*, classes que permitem a interconversão de tipos de dados. O *Frontend* comunica através de objetos JSON e as bases de dados do Django que são compostas por instâncias dos seus modelos. O *Backend* realizará então a conversão necessária sempre que um cliente pretende inserir algo nas bases de dados ou obter alguma informação contida nestas.

É necessário que o *Backend* possa ter as operações CRUD (*Create, Read, Update and Delete*) para manipular a base de dados. Estas operações são concebidas através dos verbos HTTP correspondentes, ou seja: *Create* através do *POST*, *Read* através do *GET*, *Update* através do *PUT* e *Delete* através do *DELETE*.

Para tal, definiram-se quais dessas operações cada modelo teria e foram criadas no ficheiro *views.py* do projeto. Cada modelo tem então:

Tabela 1 - Permissões de cada modelo

	GET	POST	PUT	DELETE
Grau	X			
Técnico e Posto	X	X	X	
Evento	X	X	X	X
Ambulância, Moto, Veículo e Helicóptero	X	X	X	

Quanto ao Grau, visto que este tem 4 possíveis valores e são adicionados automaticamente aquando da criação da base de dados, não precisa de ter capacidade de criar, atualizar nem eliminar.

O Técnico, Posto e as 4 subclasses do Meio de Socorro têm leitura, criação e atualização como operações possíveis. A eliminação foi interdita pois esta pode ter efeitos *cascade* que eliminem eventos existentes, o que pode ser algo indesejado.

O Evento, ao contrário dos anteriores, também tem a operação de eliminação. A eliminação de dados deve ser feita com muita cautela e evitada se for possível, mas neste caso, como o Evento é o modelo "central" da base de dados, no sentido de depender das *ForeignKeys* dos outros, a eliminação esporádica de um Evento individual pode ser conseguida sem afetar os dados dos restantes modelos.

As bases de dados que contém este tipo de informação sensível costumam ser mantidas por bastante tempo. Contudo, pode identificar-se desde já uma possível futura melhoria à aplicação, onde para proceder à eliminação, por exemplo, de um meio de socorro já obsoleto da base de dados, se teria de alguma forma de garantir a integridade dos eventos que este tivesse realizado. Isto poderia ser alcançado através de um atributo booleano novo, por exemplo "serviço", que caso fosse verdadeiro o meio estaria em funções, se fosse falso estaria obsoleto.

Frontend

Para o *Frontend*, foram reaproveitados alguns métodos desenvolvidos para as classes em Java do trabalho anterior, sendo necessário adaptá-los à linguagem Python, com as variáveis existentes na comunicação com o *Backend*.

O desenvolvimento do *Frontend* pode ser equiparado à criação de um site com o qual o utilizador vai interagir. Com este intuito foi criada uma interface com diversas funções básicas, nomeadamente, um registo dos últimos eventos para que possam ser acedidos com mais facilidade, estatísticas gerais que podem ser visualizadas na própria interface, diversos tipos de logins, como administrador que permite não só a visualização mas a utilização dos métodos CRUD definidos anteriormente, podendo este ser visto como a peça central que permite todo o tipo de operações, como técnico, que permite a visualização de informação referente ao técnico, como a sua identificação, e como meio, que permite visualizar toda a informação referente a este, permitindo também a desmobilização do meio caso esteja a ocupado a responder a um evento, e a verificação da informação do mesmo nas mesmas condições.

Além disto também é permitida a procura de informação, sendo possível obter uma lista com todos os meios, podendo definir se se quer procurar pelo tipo de meios ou pelo seu estado (disponível ou indisponível). Esta lista também conterá a informação

referentes suas informações estatísticas. Outra possibilidade é procurar um posto em específico e observar a informação deste posto, como por exemplo os meios que lhe pertencem.

Conclusão

De uma forma geral, as críticas que se punham ao modelo anterior ainda se podem colocar a este, nomeadamente devido à dificuldade relativamente elevada, ou à quantidade de tempo necessária para a implementação de funcionalidades extra (como por exemplo, a delimitação de estradas para ter em conta a quantidade de tempo e a distância percorrida pelos veículos terrestres).

Nesta segunda fase do trabalho em concreto, a maior dificuldade enfrentada foi de facto a da criação das páginas da aplicação e da sua comunicação com o *Backend*. Como se tratou do primeiro desenvolvimento de uma aplicação desta natureza, a arquitetura do projeto foi também algo difícil de definir. A arquitetura final acabou por ser decidida à medida que as dificuldades e os novos problemas foram surgindo. Contudo, agora com uma visão mais clara, algumas alterações poderiam ser feitas, como a passagem de alguns métodos do *Frontend* para o *Backend*.

Outras funcionalidades que também não foram conseguidas foi o sistema de criação de perfis e autenticação, com os utilizadores pertencendo a grupos diferentes com permissões diferentes (como administrador, técnico e meio de socorro), tendo, no lugar desta, sido utilizado o sistema *default* de administrador do Django, para a criação de perfis, e fornecendo todas as permissões a todos os utilizadores. Também, nos métodos estatísticos, não foram criados métodos que tivessem em conta a data (ano, mês e dia), que poderiam ser implementados tanto a um nível global (de todo o sistema) como individual (por técnico e meio de socorro).