EP3 de MAC350

Essa etapa foi dividida em dois desafios:

- generalizar a classe Usuário e Paciente, criando a classe Pessoa;
- desenvolver a interface gráfica para inserção, atualização, eliminação e consulta das entidades Usuário, Perfil, Serviço e Exame.

Autores: Wander Souza (10737011), Rodrigo Orem (8921590)

Quickstart

Para instalar e rodar o projeto, faça o seguinte:

```
cd src/
python3 -m venv venv
venv/bin/pip install -r requirements.txt
venv/bin/python manage.py runserver
```

Note que é necessário ter uma conexão com o banco de dados, que está configurado para usar o banco em data.ime.usp.br no arquivo `src/.env`.

Integração com o Django

Antes de começar as atividades requisitadas para o EP, configuramos nosso ambiente de desenvolvimento para rodar o projeto e se integrar com nosso banco de dados.

Parametrização flexível

Como temos dois usuários no banco de dados e por questões de segurança, é indesejável que o código no controle de versão contenha usuário e senha do banco de dados.

Para resolver isso usamos a biblioteca `dotenv`, que lê um arquivo de configuração `.env` na raiz do projeto em Django e configuramos o `settings.py` para os detalhes de conexão com o Postgre ler desse arquivo. Esse arquivo ficava no `.gitignore`, então fica fora do controle de versão. Dessa forma, o Wander tinha o seu `.env` com seus detalhes de conexão enquanto o Rodrigo usava os seus detalhes. No projeto entregue já está disponível um arquivo com os detalhes necessários para conexão.

Como ambos os membros da dupla necessitavam constantemente realizar alterações no banco, cada um com seu respectivo user, inserimos no script de criação das tabelas um trecho no qual concede todos os privilégios necessários.

Divisão por schema

Como começamos a desenvolver a terceira fase antes que a segunda fosse corrigida, notamos a necessidade de isolar os modelos físicos para que as consultas da segunda fase continuassem funcionando, mesmo com *breaking changes*.

Para conseguir isso, criamos um schema chamado `laboratorio`, usado pelo nosso sistema, enquanto o schema `public` continua atendendo às especificações da segunda etapa.

Mapeamento do modelo físico

Precisávamos então criar o `models.py`, e para facilitar o processo, usamos o comando `python manage.py inspectdb` como ponto de partida. Esse comando usa engenharia reversa para escrever o arquivo de modelos a partir das definições do banco de dados. Com isso, fomos fazendo alterações e customizações a fim de que o sistema se comportasse como desejado.

Entre as várias mudanças, trocamos os campos da tabela definindo `campo_id` ao invés de `id_campo`. Isso não era estritamente necessário, mas como se trata de um padrão que o Django usa, e que não haveria nenhuma perda significativa – apenas o ganho marginal de não ter que especificar o id em cada classe, optamos por essa modificação.

Nós seguimos usando o script de criação e inserção em SQL, que podem ser encontrados em `dist/`, conforme descrito no segundo relatório. Os scripts passaram por alterações tanto para implementar as alterações solicitadas neste EP, quanto para corrigir alguns enganos cometidos na fase anterior, graças ao feedback recebido.

Usamos também as migrations pelo Django, mas apenas para criar as tabelas de uso interno do Django, como a tabela de usuários do admin, sessões, etc. Nossas tabelas do projeto usam a opção `managed=True` para que o Django não gerencie seu ciclo de vida, nos dando maior controle sobre nosso modelo físico via SQL. Isso também permite bootstraping fácil e ágil, o que se mostrou bastante útil durante o desenvolvimento, porque eventuais iterações no modelo eram facilmente aplicadas e as tabelas eram automaticamente populadas com os scripts encontrados em `dist/`.

Desenvolvimento da aplicação

Para inserção e atualização dos registros, usamos o próprio admin do Django, pois é uma ferramenta poderosa, ideal para CRUD simples em tabelas. Foi necessário configurar no `admin.py` diversas opções para que a exibição das tabelas se comportasse de um jeito eficiente.

Por exemplo, em `usuario`, queríamos exibir o nome do tutor, que é acessível através de um autorelacionamento. Criamos então funções que manipulam os objetos e retornam atributos obtidos com o relacionamento, depois configuramos a atributo `list_display` para chamar essas funções e obter o valor necessário. O resultado é uma visualização em tabela mais rica e com informações mais úteis para o operador do sistema.

Para as consultas sofisticadas, desenhamos um wireframe no Figma (encontrado em `model/dashboard.fig`) para orientar o design e a criação do layout. Logo em seguida implementamos uma visualização em formato de dashboard, na qual cada dash representa uma consulta sobre algum aspecto do sistema. Optamos por essa visualização pois acreditamos ser uma forma bem limpa e direta de mostrar rapidamente ao usuário informações relevantes sobre

o laboratório.

O dashboard encontra-se dentro do admin, desta forma não é visível para usuários do Django não autenticados. Ao entrar na home do site, o usuário é redirecionado para o admin. O dashboard é exibido como um item dentro do admin, de maneira integrada.

Metodologia de desenvolvimento

Para desenvolvermos o projeto, criamos um repositório no GitHub e adotamos o esquema de issues e pull requests. Toda funcionalidade ou correção necessária no sistema era descrita numa issue, e cada membro da dupla escolhia em qual iria trabalhar. Ao terminar a tarefa, a pull request era submetida, e então era necessário a aprovação do outro membro para que as mudanças fossem incorporadas ao projeto final. Caso fosse identifcada alguma falha, o membro revisor fazia um review solicitando correções e apenas após isso o pull request era incorporado a branch principal.

Para mantermos o estilo do código coeso, também configuramos um linter no repositório, que não permitia que um Pull Request fosse aprovado se não atendesse a certas regras de estilo.

Jornada de usuário

Para testar nosso projeto, simulamos uma jornada de usuario, descrita a seguir: Os pacientes João e Julia vão até o laboratório para fazer exames. João quer fazer uma peniscopia e Julia quer fazer um PCR para sars-cov-2. João fez Exame com o JEF, que é administrador do sistema, e o exame ficou pronto no dia 21/05 as 10:30:45. Já julia foi atendida por Eliot, tutelado do Jef (que só pode visualizar e solicitar).