Projeto de Programação Orientada a Objetos

1st Gabriel Lopes
Aluno de Engenharia de Sistemas
Universidade Federal de MinasGerais)
Belo Horizonte, Minas Gerais

2st Kauan Valente
Aluno de Engenharia de Sistemas
Universidade Federal de MinasGerais)
Belo Horizonte, Minas Gerais

3rd Lucas Petronilho
Aluno de Engenharia de Sistemas
Universidade Federal de MinasGerais)
Belo Horizonte, Minas Gerais

Este documento escrito se trata de uma breve apresentação de um projeto desenvolvido para a disciplina de Programação Orientada a Objetos. Nosso trabalho e estudos são voltados para a construção de um software/rede social de avaliação de músicas e ábuns.

Palavras-Chave: Programação; Python; Banco de Dados Not Only/NoSQL.

I. Introdução

Nosso trabalho partiu da busca dos integrantes de se criar uma aplicação que relacionasse os interesses pessoais de cada um (nesse caso, música) às funcionalidades aplicáveis desenvolvidas ao longo do semestre na disciplina de Programação Orientada a Objetos. Inicialmente optaríamos por fazer uma rede-social voltada exclusivamente para a base de fãs de uma artista em específico. Contudo, no desenvolvimento do projeto, notamos que seria, além de viável, útil, trabalhar na construção de um espaço de avaliação musical, no geral. A ideia também partiu do pressuposto pessoal dos desenvolvedores, visto que dois dos integrantes do grupo se inspiraram em uma rede social já existente para a construção do projeto em questão. O site em questão, porém, se faz da avaliação de longas e curta-metragens, mais voltado ao cinema, ao invés da avaliação de músicas.

Nessas vias, nós decidimos que faríamos a manutenção e as bases de nosso programa utilizando Python, linguagem que foi apresentada na disciplina, alicerçada à ferramenta de banco de dados, MangoDB, que posssui especificamente sua biblioteca para uso na linguagem (Pymongo). Além disso, como era esperado, a condução do projeto foi toda conduzida pelos alicerces da Programação Orientada a Objetos, o que, de fato, facilitou em muitas vias a finalização dele.

II. Metodologia

A. Bancos de Dados

Para entendermos do que se trata o MangoDB, precisamos partir do que são bancos de dado no geral, o que é uma estrutura SQL e em que se diferencia uma ferramenta NoSQL. Segundo Silberschatz, Korth e Sudarshan, [1], os bancos de dados surgiram como uma resposta a uma série de problemas os quais eram persistentes em sistemas de arquivamento mais simples. Com o avanço da internet e da conectividade das coias, tornou-se essencial se livrar de erros causados por redundância e inconsistência nos resultados, dificuldade no acesso à dados específicos, problemas de segurança e etc. Nesse contexto, como resposta, começaram-se a ser desenvolvidos os bancos de dados. A partir daí, o dado começa a ser visto com uma visão *abstrata*. Ou seja, o banco surge como uma resposta facilitadora da análise de um número exorbitante de informações. Assim uma pessoa, abstraída da nossa realidade, torna-se apenas um objeto com dados a serem armazenados: nome, idade, gênero, nacionalidade, etc. Isso corta o fluxo do arquivo especificamente e abre alas para uma grande evolução no que tange ao controle da informação. Essas mudanças se tornam essenciais, porque fazem mais viável a criação de métodos e ações num dia-a-dia cada vez mais complexo e rodeado de novas informações. A manutenção e o fluxo desses novos espaços on-line e, essencialmente abstratos, torna-se para além do possível, essencial. Segundo os próprios autores, quanto mais abstrato, em níveis lógicos, o banco e suas ferramentas se tornam, menos complexas são as estruturas de processamento e estudo dele. Isso dialoda explicitamente com a disciplina de POO e os resultados da aplicação da metodologia dos bancos de dados se deu pela forma fluída com a qual nosso projeto foi desenvolvido. [1]

Complementar a isso, temos os estudos de Garcia-Molina, Jeffrey D. Ullman, e Jennifer Widom que definem melhor o que seria um banco de dados. Para os estudiosos, nada mais é do que uma ”coleção de informações que existem sob um longo período de tempo” [2], geralmente controladas por um sistema de gerenciamento de dados. Exemplos comuns entre as duas bibliografias citadas sugerem que os primeiros bancos vieram da necessidade em sistemas bancários, de aeroportos e para histórico financeiro de grandes corporações. Na evolução dessas tecnologias, os sistemas e dados começaram a ser desenvolvidos de várias maneiras e, até hoje, são muitas as formas de se entender e escolher qual a melhor abordagem para a administração de seus dados. O que mais conhecemos e ouvimos falar na contemporaneidade são os ”bancos de dados relacionais” [2]. Desenvolvidos na década de 1960 por Ted Codd, essa metodologia entrou em voga e se torna uma norma. Nesses casos, a distribuição de dados ocorre, como o próprio nome indica, relacionando tabelas com linhas e colunas, sendo a primeira a entidade específica daquele dado e a coluna seu atributo. O SQL ou *Structured Query Language* é a liguagem padrão para a análise e controle desses bancos, no geral. Contudo, a rigidez desses modelos de dados relacionais por vezes, pode não ser eficiente e, foi exatamente o cenário que o nosso projeto se enquadrava em seus primeiros dias de desenvolvimento, o que nos fez expadir os horiozontes na

busca de um novo sistema de gerenciamento.

B. Porque o não relacional?

É interessante como essas novas abordagens emergem mais na atualidade pois até mesmo os estudos e pesquisas sobre esse tipo específico de noção com dados são mais recentes. Pramod J. Sadalage e Martin Fowler tiveram seu trabalho publicado em 2012 e Martin Kleppmann em 2017, por exemplo. Essas tendências nos mostram que um banco de dados com maior fluidez é de muita importância para o controle do grande volume de informação que trafega constanemente no espaço virtual nos dias presentes. Bem, o não relacional representa essencialmente essa maneira mais ”aberta” de armazenamento de dados.Há, claro, outros benefícios por trás do Not Only/No SQL. Além da flexibilidade temos, preço, pois licenças desse tipo de banco tendem a ser *opem source*, a escalabilidade, que é a capacidade de um sistema de atender eficientemente a cargas de trabalho variáveis, adicionando ou removendo servidores conforme necessário e por fim a disponibilidade, visto que esses bancos geralmente usam servidores de baixo custo que estão disponíveis para possíveis problemas e frentes de trabalho. [4];

C. MangoDB e Pymongo

Com isso, recorremos ao MangoDB, ferramenta que se enquadra muito bem nas características descritas acima. Ele é um sistema de gerenciamento *open source*, capaz de armazenar um tamanho considerável de informação. Também usando a tecnologia de múltiplos servidores, ele garante uma alta performance e possui ferramentas poderosas como consultas *ad hoc*, agregação e replicação. Dentre essas, a primeira foi a mais utilizada no nosso trabalho, se tratando, basicamente, de consultas dinâmicas que podem ser criadas e executadas conforme necessário para recuperar dados específicos do banco. Isso, é claro, torna o manuseio dos dados muito mais fluído, horizontal e flexível. Esse tipo de perspeciva foi de encontro com o que nós nos propomos a fazer. Pesquisar dentro de um grande volume de informações, álbuns específicos que possuem níveis aleatórios de faixas, vários comentários diferentes, feitos inúmeras vezes sobre músicas e LP’s distintos entre si, usuários interagindo com convites de amizade, atualizando suas informações e preferidos dentro do banco, entre outros, são exemplos práticos que precisaram ser abstraídos e agregados no nosso código. Por essas questões, optamos pela feerramenta que nos serviu de forma muito conveniente na concepção desse projeto. O Pymongo seria, nesse caso, a biblioteca auxiliar do MangoDb no Python e permitiu o desenvolvimento do código. Através dela nós conseguimos extrair funções de pesquisa, alterações no banco, indexação e filtragem de dados que serão, mais a frente, explicítados aqui. Válido ressaltar que o MangoDB e Python dialogam muito bem porque os dados nesse sistema de gerenciamento, são salvos em documentos no formato JSON.

D. A Programação Orientada a Objetos nesse contexto

Nosso projeto esteve majoritariamente, senão completamente, atrelado à POO. Além do banco de dados, precisávamos garantir um fluxo adequado dos métodos do projeto e a forma com a qual lidaríamos com os dados. No escopo de metodologia comprovaremos isso com nossa estrutura de classes e suas relações, exemplos práticos do código como o uso de classes abstratas e herânça e, claro, embasamento teórico que comprove a existência de seus quatro pilares no nosso projeto. Partindo disso, é importante ressaltar o que é a POO e quais características dela são marcantes em um código alinhado a ela. Brett D. McLaughlin, Gary Pollice e David West conceberam um livro de teor mais introdutório que nos facilita essa descrição. Para os autores, a Programação Orientada a Objetos é feita através da construção de objetos, ou como utilizamos na prática, *classes*, que modelam as partes previamente planejadas de um software. [5]. Esses objetos, a partir de uma certa independência entre eles, dialogam entre si através de métodos e atributos que, com suas particularidades, formam um conjunto coeso e funcional.A POO foca na criação de software modular, extensível e por conseguinte, reutilizável.

Nosso trabalho foi idealizado pelos três integrantes e dividido, logo no início, em porções iguais para cada um. Muito alinhado a questão do que alguns autores chamam de ”decomposição orientada a objeto” [**?**] Já colocando em prática a questão da modularização, nós pensamos em três partes essenciais para o desenvolvimento do projeto: avaliação, usuários e administração de músicas e álbuns no banco de dados. Apesar de não termos trabalhado com uma metodologia ou framework específico, os afazeres fluíram de maneira bastante linear, com reuniões semanais ou quinzenais que, normalmente, indicavam entregas e apresentações do progresso no código.

Utilizando os conceitos-chave da Programação Orientada a Objetos, encapsulamento, herança, polimorfismo e abstração, nós obtivemos vias de construir os mais variados e complexos sistemas, e esses pilares, como era de se esperar, guiaram firmemente a metodologia. Sendo desejável que sejam expostos alguns exemplos e resultados de nosso trabalho, primeiro, devemos nos atentar à suas definições e utilidades.

Abstração: o primordial do banco de dados em nosso trabalho. Esse princípio nos confirma uma necessidade, senão uma funcionalidade, para que se desenvolva o código com uma melhor percepção do que se deseja. O ato de abstrair é, basicamente, entender um objeto físico ou real (um álbum, música ou usuário) para um campo virtual no qual uma máquina possa computá-la (como fizemos). A base desse primeiro fundamento é quase que filosófica e, é de suma importância para o bom desenvolvimento do trabalho. Wulf [7] complementa dizendo que, para lidar com a complexidade, nós, enquanto seres humanos, abstraímos a realidade, ignorando detalhes não essenciais e focamos em modelos generalizados e idealizados. Através da abstração, usamos pedaços de informação com maior conteúdo semântico, daí o agrupamento em objetos separados, em funcionalidades distintas e seus atributos desenhados na UML. No contexto da visão orientada a objetos, eles representam agrupamentos densos e coesos de informações, sendo abstrações de entidades do mundo real.

Encapsulamento: De maneira direta, simples e robusta, o encapsulamento diz respeito à nossa capacidade, enquanto desenvolvedores, de escolher o caráter de privacidade de um método ou atributo para dados objetos. Esse caráter pode ser definido em três tipos específicos: privado, público e protegido. Para Booch, Engle e outros estudiosos, esse pilar é complementar à abstração. Enquanto ela foca no comportamento observável, visível de um objeto, o encapsulamento se faz na implementação que gera esse objeto. Ele normalmente é alcançado, como já citado, com o ocultamento de informações da classe que não necessariamente contribuem para suas características essenciais, incluindo a estrutura e a implementação dos seus métodos. [6]

Este princípio assegura que nenhuma parte de um sistema complexo dependa dos detalhes internos de outra parte. Enquanto a abstração facilita o entendimento do que está sendo feito, o encapsulamento permite que mudanças no programa sejam feitas de forma confiável com esforço limitado. Em nosso programa esse exemplo é claro pois, buscamos seguir os bons costumes da programação e definimos os atributos gerais como privados para a maioria das classes. No escopo do trabalho, a grande ”conversa” entre as partes do ”todo” acontece majoritariamente nos códigos de interface, o que, dessa maneira, nos fez trabalhar com caráter público para elas.

-**Polimorfismo**: para que possamos entender o que é polimorfismo, é válido ressaltar primeiro o conceito de *Typing*.Em POO, ”tipagem” implica na aplicação rigorosa das classes dos objetos, de modo que objetos de tipos diferentes não possam ser intercambiados livremente, ou apenas em formas restritas. Quando ela acontece de forma dinâmica, ou seja, a compatibilidade dos tipos de dados são verificados conforme o programa é executado, temos o polimorfismo. Ele nos permite que um único nome (como uma declaração de variável) possa se referir a objetos de diversas classes que compartilham uma superclasse comum. Qualquer objeto referido por esse nome é capaz de responder a um conjunto comum de operações. Em nosso trabalho esse tipo de ação pode ser visto nas classes de interface filhas de uma superclasse abstrata que possuem métodos de mesmo nome, mas execuções diferentes; [6]

-**Herança**: Uma outra maneira de aumentar o conteúdo semântico, isto é. a variedade de informações contidas no nosso código, é reconhecendo explicitamente as hierarquias de classes e objetos dentro de sistemas de software mais complexos. A estrutura de objetos é importante porque mostra como diferentes classes colaboram através de padrões de interação, chamados de mecanismos. A estrutura de classes também é crucial, pois destaca a *hierarquia* e os comportamentos comuns dentro de um sistema. [6] Citando um exemplo mais descontraído, se quisermos estudar como funciona uma célula respiratória, estudamos primeiro células animais no geral pois as duas com certeza terão características e funções em comum. Embora tratemos cada instância de um tipo particular de objeto como distinta, assumimos que compartilham o mesmo comportamento de todas as outras instâncias desse tipo. Em nosso trabalho o que podemos citar de hierarquia facilmente, seriam as classes de interface. Uma interface principal abstrata possuí funções de exibições não definidas que se alteram de acordo com o menu seguinte. O menu de avaliações de álbum possui textos, número de opções e funcionalidades diferentes do menu de avaliações de músicas. Contudo ambos ainda são filhos da classe ”InterfaceAvaliacao()” e, sobretudo, são interfaces de menu.

E. Atividades iniciais

Um dos primeiros passos quando nos deparamos com os desafios no desenvolvimento de software, é sua arquitetura. Isto é, quais objetos estarão inclusos naquele sistema e como eles se relacionam? De que maneira suas características e funções, ou melhor dizendo, atributos e métodos, respondem, interagem e criam o resultado esperado? A chave principal para a superação desse empecilho é o que chamamos de Diagrama UML ou *Unified Modeling Language*. Trata-se de um tipo de esquema visual que facilita no entendimento do todo por simbologias uniformizadas e compreendidas amplamente por estudiosos e profissionais inseridos na área do desenvolvimento. Dessa forma, torna-se essencial que o nosso projeto tenha sido guiado por um documento dessa natureza, disponível para acesso, inclusive, no repositório remoto do projeto.

F. Idiossincrasias no nosso código

Um outro ponto muito importante para a criação desse projeto foi, claro, a configuração e criação do banco de dados, suas coleções e importação de músicas, álbuns e artistas. Para a criação do banco, foi necessário uma série de passos fundamentais ao MangoDB com o estabelecimento da conexão através de um identificador uniforme de recursos e as credenciais de usuário que garantem um acesso de leitura e edição dos dados desse banco. No arquivo ”connection.py” por exemplo, recebemos um dicionário com usuário e senha para acesso ao banco, estabelecemos uma conexão e instanciamos um *getter* que retorna somente a conexão. Essa funcionalidade foi extremamente útil em todas as classes e arquivos do código — com exceção as interfaces — pois, com o MangoDB, para modificarmos as ”Coleções”, o que seria o equivalente a uma Table em um banco relacional, utilizávamos sempre esse getter. Na conclusão do trabalho, ficamos com as coleções ”Albuns”, ”Avaliacao”, ”Comentarios”, ”Musica” e ”User”, cada uma com um papel importante para a execução do programa e com um formato específico de dicionário para armazenamento. Esse formato, dado a já citada natureza fluída do banco, pode mudar de acordo com as demandas (por exemplo, o vetor ”musicas curtidas” em um objeto aleatório da coleção ”User” pode ser alterado rapidamente, dependendo da ação executa]).

Um outro fator para as movimentações iniciais para o nosso banco foi a questão da importação de informações como título de música, artista, produtores, etc. Apesar do tamanho relativamente pequeno do nosso banco, ainda sim, temos um considerável número de informações que foram adicionadas. principalmente, por vias de leituras de um arquivo *.excel* e

com a biblioteca Pandas. A planilha foi construída de forma manual, tendo como principal fonte serviços de *streaming* de música que possuem informações específicas para cada música. O código nessa etapa extrai os dados do arquivo e os qualifica da maneira desejada em nosso banco.

Já na execução de atividades de avaliações e interações entre usuários, a principal ferramenta por trás delas foi a capacidade de *pesquisar* as informações no banco com mais facilidade, obtendo de forma direta o dicionário desejado, independentemente de sua coleção. Utilizando uma biblioteca chamada *fuzzywuzzy*, construímos uma classe *Search()* capaz de retornar resultados do banco de acordo, com uma margem de erro para digitação. Isso porque a biblioteca calcula uma proximidade entre stirngs diferentes, funcionando como uma ótima forma de tratamento de erros.

Pensamos ser válido também, apenas pincelar nesse excerto, uma discussão sobre a natureza do nosso projeto. Quando se para pra pensar que, a partir do momento em que o usuário se conecta a um banco de dados e a outras pessoas; quando que ela ou ele executa o aplicativo e faz o login, independente de seu dispositivo (desde que seja um desktop), contextos e exemplos similares podem surgir à cabeça. É interessante e até um pouco cômico pensar que fizemos uma rede-social, em esclas pequenas, claro. É possível pensar nisso visto que possuímos, para além da avaliação, a integração entre usuários. Pedidos de amizade, verificar curtidas alheias, comentários e até um status no seu perfil são opções disponíveis no Albumatic. Um projeto que visava apenas a avaliação de produtos musicais ganhou potencial para aderir a interação entre amantes de som.

Outro destaque a ser citado é o "coração" do nosso código, a Interface principal, responsável por gerenciar a interação do usuário através de menus e interfaces distintas, facilitando a navegação e a execução de operações específicas conforme as escolhas do usuário. A grande chave de seu funcionamento é dada através de mensagens, que são passadas de acordo com o que está armazenado em seu atributo "self.next". Dessa forma, quando ele assume um valor específico, uma função de exibição de outra interface é chamada! Essa foi a maneira a qual um dos integrantes do grupo conseguiu de fazer um menu principal rodar sem a necessidade de sair mais de uma vez de um *loop* eterno. Além disso, o integrante que a desenvolveu foi muito solícito ao repassar o funcionamento de sua *main* para que houvesse interação. assim, com as interfaces do restante do grupo.

III. Resultados

Os resultados foram muito gratificantes e relativamente de acordo com o esperado. Nós conseguimos desenvolver um programa executável (indicações de instalação no arquivo intruções.txt no repositório) que, para uma primeira versão, não possui muitos bugs ou problemas de funcionamento. Em geral, os três integrantes trabalharam muito bem entre si, sempre se entendendo, respeitando o tempo alheio ao mesmo tempo que se motivavam a dar continuidade nos afazeres do projeto. A metodologia desse rápido encerto se tratou de pesquisas constantes na bibliografia e no código do projeto e, rendeu, esperançosamente, bons resultados também.

Os menus estão respondendo como esperado e suas ações mudam rapidamente os dados do banco quando necessários!

IV. Conclusões

Como conclusões reiteramos a validade do MangoDB enquanto uma poderosa ferramenta de gerenciamento de banco de dados. Nós três integrantes do grupo possuíamos uma certa familiaridade com lógica de programação mas só com esse trabalho e com essa disciplina que tivemos o primeiro contato com Python. A sintaxe, em si, é muito intuitiva e, ao meu ver, é como se ela fosse a representação lógica do que pensamos quando idealizamos uma linguagem de programação. Imagino que meus colegas de trabalho pensaram o mesmo pois foram muitos comentários ressaltando a sua facilidade e ergonomia quando comparadas a outras linguagens. Como já afirmado antes, o trabalho andou sem muitos percalços, salvo uma ou duas noites em claro e alguns finais de semana em casa. Foi, para além de uma experiência engrandecedora, divertido trabalhar com amigos em algo que era do interesse dos três. Esperamos que o usuário goste e faça muito seu uso.

References

[1] KORTH, Henry F., SILBERSCHATZ, Abraham e SUDARSHAN, S. Concepções para Sistemas de Banco de Dados. Database System Concepts. 4ª edição, *S.L.*: The McGrawHill 1 Companies, 2001. pp.

[2] GARCIA-MOLINA, Hector, ULLMAN, Jeffrey e WIDOM, Jennifer. Database Systems: The Complete Book. 2ª Edição. Nova Jersey: Pearson Prentice Hall, 2002.

[3] GARCIA-MOLINA, Hector, ULLMAN, Jeffrey e WIDOM, Jennifer. Database Systems: The Complete Book. 2ª Edição. Nova Jersey: Pearson Prentice Hall, 2002. pp.40-45.

[4] MCLAUGHLINBrett D., McLaughlin, POLLICE, Gary, WEST, David. Head First Object-Oriented Analysis and Design. 1ª Edição. California: O'Reilly Media Inc, 2007.

[5] SULLIVAN, Dan. NoSQL for mere mortals. 1ª edição, Michigan: Addison-Wesley, 2015.

[6] BOOCH, Grady; MAKSIMCHUK, Robert A.; ENGLE, Michael W.; YOUNG, Bobbi J., PhD.; CONALLEN, Jim. Object-Oriented Analysis and Design with Applications. 3. ed. Boston: Pearson Education do Estados Unidos, 2007.

[7] Shaw, M. (ed.). 1981. ALPHARD: Form and Content. New York, NY: Springer-Verlag, p. 6.