Ciência de Dados e I.A. - 6° Período
Escola de Matemática Aplicada
Fundação Getúlio Vargas

Enganharia de Software

FGV Task & Bite

App de entregas na FGV

Alunos: Lavínia Dias, Isabela Yabe, Ramyro, Rodrigo Kalil
Professor: Rafael Pinho
Escola de Matemática Aplicada, FGV/EMAp
Rio de Janeiro - RJ.

Rio de Janeiro, 2024

Conteúdo

1 Classes python

Fizemos o backend do nosso aplicativo todo em python, para isso as seguintes classes foram implementadas.

1.1 EventManager

Está classe é a implementação do padrão de projeto Observer, ela contém eventos e seus inscritos, contém formas de notificações por evento e padrão, e formas de updates por evento. Ela é responsável por gerenciar eventos, assinates e notificações aplicando estratégias personalizadas. Quando um evento ocorre o EventManager chama a função notify que passa os dados para os assinantes do evento. Além de seguir o padrão de projeto Observer ele também utiliza o padrão strategy para definir estratégias de notificação e update.

1.2 PubNotifyStrategy

Essa é uma interface para aplicação de estratégias para gerenciar como os assinantes de um evento serão notificados, utilizando o padrão strategy. Assim podemos incrementar estratégias de notificação de forma simples, gerando classes filhas.

1.3 DefaultPubNotifyStrategy

Classe filhas (concreta) da interface PubNotifyStrategy, essa é a implementação default de notificação, simplesmente repassando os dados para o método update que todo inscrito contém.

1.4 SubUpdateStrategy

Aqui seguimos a mesma lógica da classe anterior, criando uma interface para criação de estratégias de updates personalizados, seguindo o padrão strategy, de forma modular e reutilizável.

1.5 DefaultSubUpdateStrategy

Como não sabemos como cada tipo de evento atualiza seus inscritos apenas adicionamos ao arquivo de log os dados recebidos pelo publicador.

1.6 PurchaseProductSubUpdateStrategy

Estratégia para o evento PurchaseProductEvent. Aqui atualizamos as tabelas dependentes da tabela de compra como a de usuário, pois precisa ser atualizada no budget, a tabela de vending machine, também precisa ser atualizda no budget pois ela recebeu dinheiro e a tabela de produtos, pois precisa atualizar estoque.

1.7 WithdrawSubUpdateStrategy

Estratégia para o evento de WithdrawMoneyEvent. Atualizamos a tabela de vending machine, pois o evento notifica que o proprietário retirou dinheiro da vending machine fazendo com que seja necessário a atualização do budget da vending machine.

1.8 ValidationStrategy

Interface para criarmos classes concretas de validação.

1.9 BannedWordsStrategy

Classe concreta da ValidationStrategy, essa estratégia é usada para verificar se os valores de um dicionário contém palavra proibidas e se tiver não retorna esse dado.

1.10 SQLInjectionStrategy

Classe concreta da ValidationStrategy para verificar se nenhum comando sql está sendo passado.

1.11 CustomLogger

Classe singleton para geração de logs personalizados para auxiliar na produção do código.

1.12 DatabaseManagerCentral

Classe Singleton para criação de uma única instância no sistema. Ela tem o objetivo de centralizar o controle de todas as tabelas do banco de dados, simplificar o gerenciamento das múltiplas tabelas e realizar o CRUD, as notificações de eventos e a integridade dos dados.

Essa classe é uma facade, oferece uma interface única para acesso e manipulação de várias tabelas, cada uma gerenciada por instâncias de DatabaseManager. Afim de separar a lógica de acesso ao banco de dados do restante do sistema, facilitando a manutenção e a extensibilidade.

1.13 DatabaseManager

Classe que gerencia as tabelas no banco de dados, permitindo operações de CRUD e outras ações relacionadas à manipulação de tabelas. Ela também usa o padrão de arquitetura Pub-Sub nas tabelas publicadoras e ouvintes, possibilitando que tabelas dependentes sejam notificadas quando alterações acontecem.

1.14 PasswordHasher

Classe responsável por hashear senhas, para podermos salvar de forma segura a senha dos usuários e owners.

2 Padrões de Design

2.1 Singleton

Usamos o método singleton para criarmos nossa classe de Logs para nos auxiliar no desenvolvimento do projeto de forma organizada e eficiente. Aqui tivemos o cuidado de mantermos com singleton para evitarmos múltiplas instâncias e assim evitnado mensagens de logs duplicadas, centralizarmos a lógica de logging e assim ser usado por todo o sistemas, facilitando rastreios. Também criamos um arquivo de logs, assim tudo que acontece no sistema está registrado em um único local.

Utilizamos o método singleton para criarmos a nossa interface simplificada, a nossa facade, com o objetivo dela ser a única que poderá controlar nosso banco de dados MySQL, garantindo concistência nos dados, pois evita múltiplas conexões. Também podemos centralizar nossas tabelas nela.

2.2 Facade

O Facade foi utilizado na classe DatabaseManagerCentral para que ela seja uma interface simplificada e única capaz de interagir com todos os subsistemas e componentes do nosso programa.

Nela é gerado todos os DatabaseManagers que controlam cada um uma tabela individualmente, é também gerado o EventManager responsável pelas comunicações do tipo Pub/Sub. Também é implementado métodos que resumem comunicações entre diversas classes, evitando que o usuário tenha que ter conhecimento profundo das demais classes existentes.

2.3 Decorator

O decorator foi bastante utilizado para modificarmos comportamentos sem ter que mudarmos o que já tinhamos, facilitnado a manutenção do código e a única responsabilidade, o caso mais básico foi na criação do singleton, depois nos decoradores do arquivo decorators_method, onde criamos o request_validations para verificação do que é passado para o nosso sistemas através de requisições HTTP, e também não permitir que palavras ofencivas sejam usadas. Temos também o immutable_fields que tornar colunas imutáveis numa tabela atráves do DatabaseManager, assim não temos que verificar sempre que alguém faz um update se estão tentando mudar uma coluna imutável. Também criamos um decorador para hashear senhas em tabelas que contém senhas, como a user_profile e a owners_profile, assim mais uma vez tirando a responsabilidade da classe de hashear suas colunas de senha.

2.4 Observer

Aqui para podermos usar nos familiarizarmos com esse padrão e fugir do esperado, criamos um sistema de pubsub para comunicação entre tabelas dependentes. Exemplo: quando alguém faz uma compra a tabela de compras é atualizada, porém quando essa tabela é atualizada mais três também precisam a tabela de produtos (precisamos retirar do estoque), a de usuário (retirando dinheiro de sua conta), e a

de vendas (adicionando dinheiro no seu caixa). Usamos a lógica da tabela de vendas ser um publicador e as três tabelas mencionadas assinantes, assim sempre que ela for atualizada ela manda uma notificação para seus inscritos e eles se atualizam.

2.5 Strategy

Como dito anteriormente criamos um PubSub. Criamos ele de forma genérica para podermos utilizarmos dele para diversos eventos, para isso ser possível criamos estratégias de notificações, sendo possível criar uma estratégia para cada tipo de evento criado e caso não haja uma notificação expecífica é usado a default. Da mesma forma que o padrão strategy foi usado para Pub Strategy, ele foi utilizado para o Sub Strategy.

Optamos por utilizar strategy no Validation Strategy para podermos criar diversas formas de validação, adicionando regras sem termos que modificar o código!

Padrões de projetos implementados: strategy, decorators.

3 Tabelas e Relacionamentos

3.1 Tabela `owners_profile`

Esta tabela armazena informações sobre os proprietários das máquinas de venda.

- **Atributos**: `id` (chave primária), `username`, `email`, `password`, `first_name`, `last_name`, `birthdate`, `phone_number`, `address`, `budget` (valor arrecadado nas máquinas pelo dono).

3.2 Tabela `users_profile`

Armazena informações sobre os usuários que interagem com as máquinas de venda e produtos.

- **Atributos**: `id` (chave primária), `username`, `email`, `password`, `first_name`, `last_name`, `birthdate`, `phone_number`, `address`, `budget` (saldo disponível para realizar compras).

3.3 Tabela `vending_machines_profile`

Contém informações sobre as máquinas de venda automática.

- **Atributos**: `id` (chave primária), `name`, `location` (localização detalhada da máquina), `status` (indica o estado da máquina, como ativa ou inativa), `timestamp` (data e hora de criação ou última atualização), `owner_id`.
- **Relacionamentos**: Referencia a tabela `owners_profile` através do campo `owner_id`.

3.4 Tabela products_profile

Armazena informações sobre os produtos disponíveis nas máquinas de venda.

- **Atributos**: id (chave primária), name, description, price, quantity, vending_machine_id, timestamp (data e hora de adição ou atualização do produto).
- **Relacionamentos**: Referencia a tabela vending_machines_profile através do campo vending_machine_id.

3.5 Tabela product_complaint

Registra reclamações de produtos feitas pelos usuários.

- **Atributos**: id (chave primária), text (detalhes da reclamação), product_id, user_id, timestamp (data e hora da reclamação).
- **Relacionamentos**: Referencia as tabelas products_profile e users_profile.

3.6 Tabela product_comment

Registra comentários sobre produtos feitos pelos usuários.

- **Atributos**: id (chave primária), text (conteúdo do comentário), product_id, user_id, timestamp (data e hora do comentário).
- **Relacionamentos**: Referencia as tabelas products_profile e users_profile.

3.7 Tabela purchase_transaction

Registra transações de compra realizadas nas máquinas de venda.

- **Atributos**: id (chave primária), user_id, product_id, vending_machine_id, timestamp (data e hora da transação), quantity (quantidade adquirida), amount_paid_per_unit (preço unitário pago pelo produto, podendo refletir promoções ou descontos).
- **Relacionamentos**: Referencia as tabelas users_profile, products_profile, e vending_machines_profile.

3.8 Tabela vending_machine_complaint

Registra reclamações sobre máquinas de venda feitas pelos usuários.

- **Atributos**: id (chave primária), text (detalhes da reclamação), vending_machine_id, user_id, timestamp (data e hora da reclamação).
- **Relacionamentos**: Referencia as tabelas vending_machines_profile e users_profile.

3.9 Tabela `vending_machine_comment`

Registra comentários sobre máquinas de venda feitos pelos usuários.

- **Atributos**: `id` (chave primária), `text` (conteúdo do comentário), `vending_machine_id`, `user_id`, `timestamp` (data e hora do comentário).
- **Relacionamentos**: Referencia as tabelas `vending_machines_profile` e `users_profile`.

3.10 Tabela `favorite_vending_machines`

Registra as máquinas de venda favoritas dos usuários.

- **Atributos**: `id` (chave primária), `vending_machine_id`, `user_id`.
- **Relacionamentos**: Referencia as tabelas `vending_machines_profile` e `users_profile`.

3.11 Tabela `favorite_products`

Registra os produtos favoritos dos usuários.

- **Atributos**: `id` (chave primária), `product_id`, `user_id`.
- **Relacionamentos**: Referencia as tabelas `products_profile` e `users_profile`.

4 Requisitos implementados

4.1 (RF10) Visualizar Estoque

Foi criado as tabelas Vending_Machines_Profile e Products_Profile no nosso banco de dados e as classes que as gerenciam foram instaciadas na classe facade DatabaseManagerCentral. Para a visualização do estoque foi implementado um html, assim podemos ver todos os produtos de uma loja e seus estoques.

4.2 (RF14) Perfil da VendingMachine

Foi criado a tabela Vending_Machines_Profile no nosso banco de dados e a classe que a gerencia foi instaciada na classe facade DatabaseManagerCentral. Para a sua visualização foi implementado um html para ver cada vending machine e seus comentários e reclamações também.

4.3 [NRB] Selecionar Vending Machine

Foi criado a tabela Favorite_Vending_Machine que contém coluna de usuário e a vending machine favoritada, a lógica do back-end para a adição de elementos dela também.

4.4 (NRA) Visualizar Perfil de Produto

Foi criado as tabelas Vending_Machines_Profile e Products_Profile no nosso banco de dados e as classes que as gerenciam foram instaciadas na classe facade DatabaseManagerCentral. Para a visualização do estoque foi implementado um html, assim podemos ver todos os produtos de uma loja.

4.5 (RF07) Comentar sobre Produto

Foi criado a tabela Product_comment que contém coluna de usuário e o comentário, a lógica do back-end para a adição de elementos dela também.

4.6 (RF20) Relatórios Básicos

Começamos a implementação no back-end com uma classe que lê arquivos de logs para capturar tudo o que acontece dentro do sistema, e uma chamada SQL para visualizar vendas dentro de um período de tempo.

4.7 (RF33) Reportar Problemas

Para a vending machine é possível fazer reclamações na mesma página de onde podemos comentar e visualizar vending machines.

4.8 Login

Criamos uma classe para hashear senhas as guardando no banco de dados como um hash, e para o login encriptografamos a senha para o hash e fazemos o match.

4.9 Transações de vendas

Criamos um método de transação de vendas usando o pubsub, assim quando alguém realiza uma compra o estoque é reduzido, o dinheiro do cliente também e o caixa da máquina recebe o valor.

4.10 Logs

Criamos uma classe de logs, ela foi utilizada em todo o sistema e um arquivo registra os logs de info, warning e error. Assim podendo ser utilizado caso necessário.

4.11 Validação no POST

Criamos uma classe para validar o que seria colocado num comentário ou reclamação para nos previnirmos de ataques e poupar nossos clientes de outros clientes que utilizam palavras de baixo calão.

5 Testes

Durante toda a implementação do sistemas foi utilizado testes usando MagicMock para simularmos comportamentos de bancos de dados reais, porém após a implementação de todo o sistema fomos testar com um banco real e algumas adaptações tiveram que ser realizadas desatualizando os testes, então alguns testes foram removidos, porém deixamos os que ainda estão atualizados.