Relatório
Gerenciamento de conta bancária

Sumário

Brainstorm:

A primeira etapa do nosso projeto foi o Brainstorm, que consistiu na filtragem de ideias, qual será a funcionalidade? Qual a melhor abordagem para codificar? Qual será a divisão de funções? Decidimos então que o back-end, ou seja, escrita de códigos será feita principalmente **por Kauan Estrela e Henrique Ramos**, porém com a ajuda dos TeamMates (Colegas de time), assim como **Alice Santos** foi encarregada de assumir o relatório geral da equipe, assim condensando todos os relatórios pessoais dos participantes, tudo isso sendo organizado por **Henrique ramos**, na etapa de PLANEJAMENTO, ele se prontificou a organizar os testes, reuniões e planilhas, assim gerando um controle e otimização de tempo, por final restou a **David Wallace** uma revisão geral do código e interface do terminal juntamente com **Leonardo Levi.**

Planejamento

Nosso Planejamento foi baseado em uma planilha Excel separada em 3 partes:

1- A parte de controle e gerenciamento de atividades para alinhar quantas atividades foram definidas e o status delas
2- A planilha de foco geral para ter uma noção geral de problemas e quais processos são prioridades e status
3- A planilha mais detalhada das atividades, com intuito de alinhar semanalmente as tarefas, definir quem foi designado para cada uma delas e qual o status de cada atividade, além de prazos e tempo gasto discutindo sobre o teste.

As planilhas de controle possui um reconhecimento por cor e palavras chaves para fácil identificação

Projeto de Gerenciamento de Conta Bancária									
Status		Pendentes		Agendado		Feitos		Atrasado	
Numero de Testes	6	Nº Testes	0	N° Testes	3	Nº Testes	2	N° Testes	1

Controle Semanal de Atividades								
Semana 1	Total	6	Agendados	0	Atrasado	1	Feitos	5
Semana 2	Total	7		2		4		1

Na primeira semana tivemos um progresso significativo onde até então tudo estava indo de acordo com o planejamento e seguindo o cronograma porem houve um atraso na codificação

Responsável	Teste	Data limite	Status
ALL	Brainstorm	25-Oct	Feito
Henrique	Planejamento	24-Oct	Feito
Kauan	Codificação	27-Oct	Atrasado
Lilice	Relatório	30-Oct	Agendado
Wallace e Levi	Interface	30-Oct	Agendado
ALL	Revisão Geral	1-Nov	Agendado

A planilha acima refere-se ao **Projeto de Gerenciamento de Conta Bancária**. Mesmo não seguindo o cronograma conseguimos controlar os atrasos devido a ternos feito o planejamento adiantado. As datas marcadas para a finalização de cada processo foi marcado em datas que nos permitisse fazer alterações em caso de atraso.

A planilha abaixo refere-se ao **Controle Semanal de Atividades**, essa planlha é mais detalhada e engloba um processo semanal de cada reunião e atividade feita

Planejamento semanal								
Data de Entre	Atividades	Responsaveis	Semana	Tempo Previsto	Tempo Real	Status	Reagendament	Status final
23-Oct	Código Base da conta bancária	Kauan e Henrique	1	indeterminado	indeterminado	feito		Concluido
23-Oct	Lista de itens do código	All	1	5 min	8 min	feito		Concluido
23-Oct	Divisão de tarefas	All	1	10 min	10 min	Feito		Concluido
24-Oct	Começo do relatório	Alice	1	5 min	6 min	feito		Concluido
26-Oct	Finalização do Código	Kauan e Henrique	1	2 horas	6 horas	Atrasado	2-Nov	Agendado
27-Oct	Reunião para Alinhamento do Planejamento	All	1	50 min	30 min	feito		Concluido
30-Oct	Revisão do código	Wallace	2	5 horas	6 horas	feito		Concluido
30-Oct	Finalização da Interface	Wallace e Levi	2	40 min	-	Atrasado		
30-Oct	Finalização do Relatório	Alice	2	30 min		Atrasado	2-Nov	Agendado
31-Oct	Sharing	All	2	55 min		Atrasado	1-Nov	Agendado
31-Oct	Reunião Final de Alinhamento para Apresentação	All	2	1h20		Atrasado	2-Nov	Agendado
1-Nov	Terminal	Wallace e Levi	2	45 min		Agendado		
2-Nov	Resolução de atrasos que houver	All	2	30 min		Agendado		

O status final foi adicionado na segunda semana de trabalho após termos alguns atrasos significativos. A ideia é reagendar a atividade que não foi concluida na data prevista e alinhar a situação após a data ser marcada.

Apesar dos atrasos todos foram concluidos na data (02/11/23).

Planejamento semanal								
Data de Entre	Atividades	Responsaveis	Semana	Tempo Previsto	Tempo Real	Status	Reagendament	Status final
23-Oct	Código Base da conta bancária	Kauan e Henrique	1	indeterminado	indeterminado	feito		Concluido
23-Oct	Lista de itens do	All	1	5 min	8 min	feito		Concluido
23-Oct	Divisão de tarefas	All	1	10 min	10 min	Feito		Concluido
24-Oct	Começo do relatório	Alice	1	5 min	6 min	feito		Concluido
26-Oct	Finalização do Código	Kauan e Henrique	1	2 horas	6 horas	Atrasado	2-Nov	Concluido
27-Oct	Reunião para Alinhamento do Planejamento	All	1	50 min	30 min	feito		Concluido
30-Oct	Revisão do código	Wallace	2	5 horas	6 horas	feito		Concluido
30-Oct	Finalização da Interface	Wallace e Levi	2	40 min	-	Atrasado	2-Nov	Concluido
30-Oct	Finalização do Relatório	Alice	2	30 min		Atrasado	2-Nov	Concluido
31-Oct	Reunião Final de Alinhamento para Apresentação	All	2	1h20		Atrasado	2-Nov	Concluido
1-Nov	Terminal	Wallace e Levi	2	45 min		Feito		Concluido
2-Nov	Resolução de atrasos que houver	All	2	30 min		feito		Concluido

Codificação (Back-end)

Back end, Kauan Estrela.

O código utiliza o módulo sqlite3 para emular o SQL em Python.

Criamos a função criar_banco() é definida para criar o banco de dados e a tabela de usuários,A função encerra a transação e fecha a conexão com o banco de dados (conn.commit() e conn.close())

Criamos a função adicionar_usuario(nome, cpf) permite adicionar um novo usuário ao banco. A função também encerra a transação e fecha a conexão com o banco de dados. (conn.commit() e conn.close())

Criamos a função consultar_saldo(cpf) consulta e exibe o saldo de um usuário com base no CPF. Se o usuário não for encontrado, uma mensagem de aviso é exibida.

Criamos a função adicionar_saldo(cpf, valor) permite adicionar saldo a uma conta de usuário.

A função sacar(cpf, valor) permite ao usuário realizar saques, desde que o saldo seja suficiente.

Criamos a função main() contém um loop que apresenta um menu de opções ao usuário.
A função main() chama a função criar_banco() para garantir que o banco de dados esteja pronto. Em seguida, começa um loop que apresenta um menu de opções para o usuário:

A opção "1" permite consultar o saldo de um usuário.

A opção "2" permite adicionar saldo a uma conta.

A opção "3" permite fazer um saque.

A opção "4" permite sair do programa.

. O loop continuará até que o usuário escolha a opção "4" para sair do programa.

O usuário pode escolher entre consultar saldo, adicionar saldo, realizar saque ou sair.

usamos o if __name__ == "__main__": main() para separar o código que deve ser executado diretamente do código que é destinado a ser reutilizado como uma biblioteca ou módulo.
(basicamente PUXAR a função main que criamos)

As 13:00 da tarde do dia 21/10/2023 tivemos uma reunião de uma hora
onde declaramos que temos que: Alterar o codigo para POO, e aplicar os seus conceitos, precisamos tambem adicionar o metodo “ Transferencia, seja por pix, TED, ou de uma conta para outra, adicionaremos tambem o sistema de cargos. Onde adicionamos o metodo scrum ao nosso planejamento

Kauan Estrela, 22/10/2023
Aprimorei o codigo e adicionei o POO, Criação do Método transferir:

Foi criado o método transferir na classe Banco para gerenciar as operações de transferência de saldo entre usuários.

Foi criado um dicionário chamado transferencias_programadas para armazenar as transferências programadas via TED. Esse dicionário permite rastrear quais transferências devem ser realizadas após um atraso de um minuto.

Atualização do Menu:

O menu da classe Banco foi atualizado para incluir novas opções, como "Transferência PIX" e "Transferência TED."

Opção "Transferência PIX":

Quando o usuário escolhe a opção "5" no menu, o código solicita os CPFs de origem e destino e o valor a ser transferido e chama o método transferir com o argumento "pix". Essa transferência é realizada imediatamente, descontando o valor da conta de origem e adicionando-o à conta de destino.

Opção "Transferência TED":

Quando o usuário escolhe a opção "6" no menu, o código solicita os CPFs de origem e destino e o valor a ser transferido. O método transferir é chamado com o argumento "ted". Nesse caso, a transferência TED é programada, e a quantidade a ser transferida é armazenada no dicionário transferencias_programadas. O código imprime uma mensagem informando que a transferência foi programada para ser realizada em um minuto.

Atraso de 1 Minuto para Transferência TED(convencionalmente demora de um a três dias, mas como é um codigo rapido usei um minuto para demonstrar):

Após programar a transferência TED, o código utiliza time.sleep(60) para aguardar 60 segundos (1 minuto) antes de realizar a transferência. Durante esse atraso, o programa fica inativo. Após o atraso, o código verifica se a transferência ainda está no dicionário transferencias_programadas e a executa, descontando o valor da conta de origem e adicionando-o à conta de destino.

o método adicionar_usuario verifica se um usuário com o mesmo CPF já existe no banco de dados e, se não existir, adiciona um novo usuário com um saldo inicial de zero.

self.cargo_atual = "cliente"

Aqui, estamos definindo um atributo chamado cargo_atual e o configurando como "cliente". Isso pode representar o nível de acesso ou o tipo de conta.

self.transferencias_programadas = {}

Este atributo cria um dicionário vazio chamado transferencias_programadas. Um dicionário é uma estrutura de dados que associa chaves a valores.

O método consultar_saldo consulta o saldo de um usuário com base no CPF fornecido.

O método adicionar_saldo adiciona um valor ao saldo de um usuário.

O método sacar retira um valor do saldo de um usuário, se houver saldo suficiente.

Por fim, o método exibir_menu cria um menu para que o usuário possa escolher qual operação deseja realizar. O usuário pode adicionar usuários, consultar saldos, adicionar saldo, sacar dinheiro, fazer transferências PIX ou TED e sair do programa.

"banco = Banco()"

Criamos um objeto da classe Banco, que inicializa o banco de dados e as configurações.

banco.exibir_menu()

Este comando chama o método exibir_menu(), que inicia o menu para o usuário interagir com o programa.

Em resumo, este código é um exemplo simples de um sistema bancário em que você pode adicionar usuários, consultar saldos, adicionar saldo, sacar dinheiro e fazer transferências. É uma demonstração básica de como um programa Python pode interagir com um banco de dados SQLite.

o proximo passo foi reformular e adicionar funções ao menu

O que antes tinha seis funções passou a ter onze, o menu atual conta com:

print("1. Criar Cadastro")

print("2. Extrato Bancário")

print("3. Depositar")

print("4. Sacar saldo")

print("5. Transferência PIX")

print("6. Transferência TED")

print("7. Realizar Pagamento")

print("8. Excluir cadastro")

print("9. Acesso Administrador ")

print("10. Reclame aqui")

print(“11. Sair”)

Em ordem, adicionamos o item 7, que consiste em realizar pagamentos de contas como agua e luz, onde o usuario escolhe o tipo de conta e realiza o pagamento

Já no item 8 do Menu, o usuario tem a opção de excluir a própria conta, usamos um sistema para induzir o usuario a sacar ou depoisitar o saldo, assim sendo impossivel excluir uma conta existindo saldo

No item 9 Temos o acesso administrador, com foco em ser um menu a parte para funcionarios do banco, tivemos bastante problema para conectar os dois menus,

No item 10, adicionamos a opção sair, que ao pé da letra ativa um Break no codigo, essa função é a porta de saida do nosso codigo.

A revisão ficou com David Wallace e Leonardo Levi, assim como a interface do terminal

Interface do terminal atualizada

removida opção adicionar administrador

removido metodo adicionar administrador

A proxima etapa foi dividir e conquistar, nós modularizamos o codigo para ficar mais facil corrigir erros e debbugar, depois importamos e conectamos metodo por metodo, e nesta etapa, não tivemos exito,
pelo fato do codigo estar todo na classe “ Banco “, modularizar se tornou impossível, o codigo é funcional, porém inseguro.