Projeto de Teste

Sessão 3

Solução de Software de TI para Negócios

Designer de Projeto de Teste Independente: Ramin Mohammaddoust

Validador de Projeto de Teste Independente: Afshin Dehghani

Introdução

Nesta sessão, você vai assumir o papel de um desenvolvedor full-stack, responsável por construir a interface do usuário (UI) front-end e a API back-end para o novo sistema de software da padaria.

O foco desta sessão é traduzir o design e os requisitos da Sessão 2 em um aplicativo de desktop funcional para a equipe e uma API robusta para gerenciar dados de produtos e pedidos. Você será avaliado em sua capacidade de criar interfaces de usuário intuitivas e visualmente atraentes, bem como em sua proficiência no desenvolvimento de APIs seguras e eficientes.

Esta sessão foi projetada para avaliar suas habilidades em:

- **Desenvolvimento Front-End:** Projetar e implementar interfaces de usuário que sejam fáceis de usar, visualmente atraentes e alinhadas com a marca Belle Croissant Lyonnais.
- **Desenvolvimento Back-End:** Criar uma API RESTful que siga as melhores práticas do setor, lide com interações de dados com segurança e forneça uma experiência perfeita para o aplicativo front-end.
- **Integração:** Garantir comunicação e troca de dados sem problemas entre os componentes front-end e back-end.
- **Solução de Problemas:** Identificar e resolver desafios técnicos que surgem durante o desenvolvimento.

Conteúdo

Este pacote de sessões inclui os seguintes materiais:

1. **Instruções da Sessão (PDF):** Instruções detalhadas descrevendo as tarefas a serem concluídas e as entregas esperadas para esta sessão.
2. **Common Folder:** Esta pasta contém recursos adicionais, como o logotipo Belle Croissant Lyonnais, ícones, guia de estilos e outros recursos de design que podem ser usados durante todo o desenvolvimento do aplicativo.
3. **Esquema de Banco de Dados (SQL):** Um script SQL que contém a estrutura das tabelas Promoções e Programa de Fidelidade, que serão utilizadas nesta sessão.

Descrição do Projeto e Tarefas

Nesta sessão, você vai criar a base para o aplicativo de desktop Belle Croissant Lyonnais.

Diretrizes:

1. **Fácil de Usar:** Tornar a interface simples e fácil de entender para a equipe.
2. **Parece Bom:** Seguir o Guia de Estilos Belle Croissant Lyonnais para o design.
3. **Funciona Bem:** Verificar se todas as partes do aplicativo funcionam corretamente e sem erros.
4. **Seguro:** Proteger os dados do cliente e garantir que o aplicativo seja seguro de usar.
5. **Pontualmente:** Concluir todas as tarefas dentro do limite de tempo.

Considerações Técnicas:

1. **Configuração do Banco de Dados:** Criar e preencher o banco de dados de acordo com o esquema fornecido.
2. **Desenvolvimento de API:** Implementar uma API RESTful que siga as práticas recomendadas e inclua autenticação.
3. **Interface de Usuário:** Desenvolver telas intuitivas para gerenciamento de produtos e pedidos.
4. **Validação de Dados:** Certifique-se de que a entrada do usuário esteja correta e completa para todas as operações.
5. **Tratamento de Erros:** Exibir mensagens claras para o usuário se houver algum problema.

Considerações Adicionais:

- O aplicativo deve funcionar sem problemas e rapidamente no ambiente de desenvolvimento fornecido.
- Usar rótulos e instruções claros para todos os elementos da interface do usuário (UI).
- Organizar as informações de uma maneira que seja fácil para a equipe entender.
- Considerar casos extremos e possíveis erros na entrada do usuário e no tratamento de dados.
- Implementar testes de unidade abrangentes para garantir a funcionalidade da API.

Instruções ao Participante

3.1 Configuração do Banco de Dados e Importação de Dados

Objetivo:

Criar e preencher o banco de dados Belle Croissant Lyonnais de acordo com o esquema fornecido, garantindo a precisão e integridade dos dados.

Tarefas:

1. **Criação de Banco de Dados:**
    - Criar um banco de dados chamado BelleCroissantLyonnais (ou um nome semelhante e relevante).
2. **Execução do Esquema:**
    - Executar o script SQL fornecido (Database_Schema.sql) para criar as seguintes tabelas e suas relações:
        - Produtos
        - Clientes
        - Ordens
        - OrderItems
3. **Importação de Dados:**

- Importar os dados dos arquivos CSV limpos (products_cleaned.csv, customers_cleaned.csv e sales_transactions_cleaned.csv) para as tabelas de banco de dados correspondentes. Verificar se os tipos de dados correspondem à definição do esquema.

Entregáveis:

- **Credenciais do Banco de Dados:** Fornecer a cadeia de conexão ou as credenciais necessárias para acessar seu banco de dados (nome do servidor, nome do banco de dados, nome de usuário, senha). Você pode armazenar essas informações em um arquivo de texto simples chamado Session3_DatabaseCredentials.txt.

Notas adicionais:

- Prestar atenção aos tipos de dados nos arquivos CSV e verificar se eles correspondem aos tipos de dados definidos no esquema.
- Usar ferramentas ou técnicas de importação apropriadas para transferir com eficiência os dados para o banco de dados.
- Verificar novamente os dados após a importação para garantir sua precisão e integridade.
- A avaliação se concentrará na exatidão e integridade da configuração do banco de dados e do processo de importação de dados.

3.2 Desenvolvimento de API de Back-End

Objetivo:

Desenvolver um API RESTful segura e eficiente usando a API Web .NET que permite que o aplicativo da área de trabalho Belle Croissant Lyonnais interaja com o banco de dados.

Tarefas:

1. **Endpoints da API:** Faça as seguintes maneiras para o aplicativo se comunicar com o banco de dados:
   - **Produtos:**
     - GET /api/products: Obter todos os produtos
     - GET /api/products/{id}: Obter um produto por ID
     - POST /api/products: Adicionar novo produto
     - PUT /api/products/{id}: Alterar produto por ID
     - DELETE /api/products/{id}: Remover produto por ID
   - **Clientes:**
     - GET /api/customers: Obter todos os clientes
     - GET /api/customers/{id}: Obter um cliente por ID
     - POST /api/customers: Adicionar novo cliente
     - PUT /api/customers/{id}: Alterar cliente por ID
   - **Ordens:**
     - GET /api/orders: Obter todos os pedidos
     - GET /api/orders/{id}: Obter um pedido por ID

- POST /api/orders: Adicionar novo pedido
- PUT /api/orders/{id}/complete: Finalizar pedido por ID
- PUT /api/orders/{id}/cancel: Cancelar pedido por ID

2. **Verificação de Dados:**
    - Certifique-se de que o aplicativo envie o tipo certo de dados para o banco de dados (números, letras, datas, etc.).
    - Limpar os dados para garantir que sejam seguros de usar.
3. **Tratamento de Erros:**
    - Enviar as mensagens certas de volta para o aplicativo se houver um problema.
4. **Autenticação:**
    - Certifique-se de que apenas as pessoas certas possam usar a API. Usar um sistema de senha (Autenticação Básica) onde o nome de usuário é "staff" e a senha é "BCLyon2024".

Entregáveis:

- **API em Execução:** Faça a API funcionar no local determinado.
- **Session3_API_Endpoints.txt:** Um arquivo listando o que a API pode fazer, como pedir coisas e o que esperar de volta (em JSON).

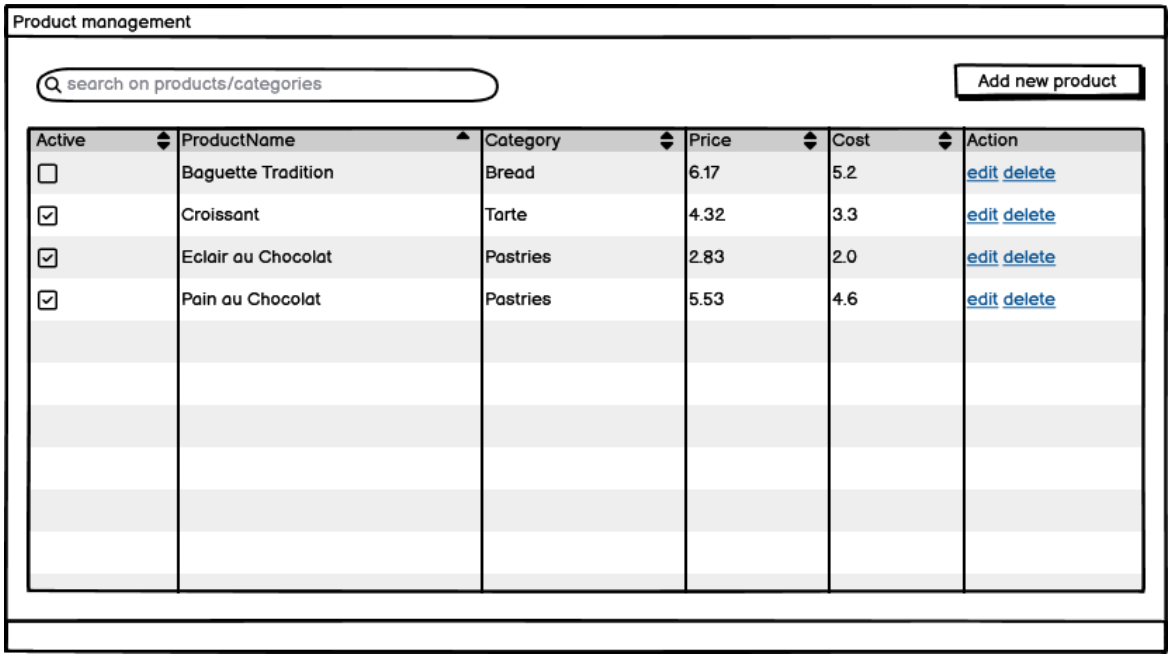
3.3 Desenvolvimento de UI - Gerenciamento de Produtos

Objetivo:

Desenvolver uma interface de desktop amigável para gerenciar produtos no aplicativo Belle Croissant Lyonnais, utilizando os endpoints da API criados na Tarefa 3.2.

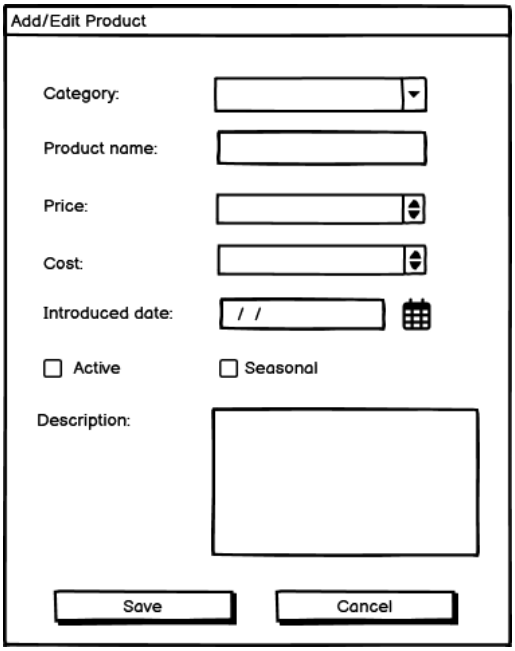
Tarefas:

1. **Produto Listado:**



- Exibir todos os produtos buscados no endpoint da API GET /api/products em uma tabela.
- Colunas da tabela: ProductName, Category, Price, Cost, Active (Sim/Não).
- Adicionar classificação (crescente/decrescente) para cada coluna.
- Adicionar uma barra de pesquisa para filtrar produtos por nome ou categoria.

2. **Adicionar/Editar Formulário do Produto:**



- Criar um formulário para adicionar novos produtos ou editar os existentes.
- Campos de formulário:

- ProductName (entrada de texto, requisitos, máximo de 100 caracteres)
- Categoria (lista suspensa, requisitos, valores de products_cleaned.csv)
- Preço (a entrada numérica, requisitos, deve ser um número positivo)
- Custo (entrada numérica, requisitos, deve ser um número positivo e menor que Preço)
- Descrição (área de texto, opcional)
- Sazonal (caixa de seleção)
- Ativo (caixa de seleção)
- IntroducedDate (seletor de data, requisitos)

  - O formulário deve ter os botões "Salvar" e "Cancelar".
  - Validar os dados de entrada.
  - Usar a API para:
    - POST /api/products para salvar novos produtos.
    - PUT /api/products/{id} para atualizar os produtos existentes.
    - Manipular as respostas da API e exiba as mensagens apropriadas (sucesso/erro) para o usuário.

3. **Deletar funcionalidade do produto:**
   - Adicionar um botão "Deletar" ao lado de cada produto na listagem.
   - Mostrar uma caixa de diálogo de confirmação antes de deletar.
   - Usar a API (DELETE /api/products/{id}) para deletar o produto.
   - Manipular a resposta da API e atualizar a lista de produtos conforme necessário.

Entregáveis:

- Incorporar a interface do usuário (UI) ao arquivo Session3_DesktopApp.exe existente.

Notas adicionais:

- Seguir as práticas recomendadas para design e desenvolvimento de interface do usuário (UI).
- A interface do usuário (UI) deve ser responsiva e se adaptar a diferentes tamanhos de tela.
- Considerar o tratamento de erros e os comentários do usuário na interface do usuário (UI).

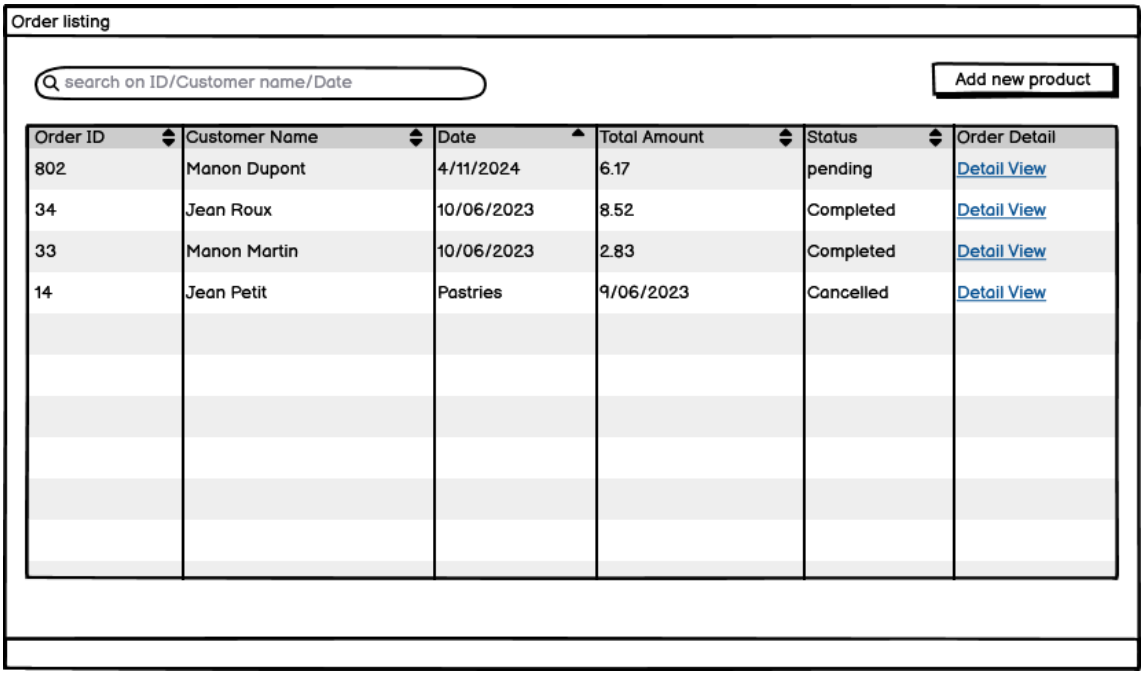
3.4 Desenvolvimento de UI - Gerenciamento de Pedidos

Objetivo:

Desenvolver uma interface de desktop amigável para gerenciar pedidos de clientes no aplicativo Belle Croissant Lyonnais, utilizando os endpoints de API criados na Tarefa 3.2.

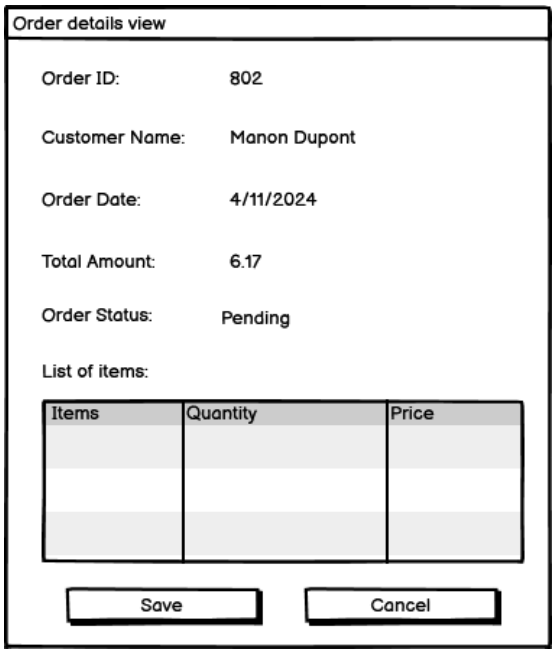
Tarefas:

1. **Lista de Pedidos:**



- Exibir todos os pedidos do banco de dados em uma tabela.
- Colunas da tabela: ID do pedido, Nome do cliente, Data, Valor total, Status (Pendente, Processando, Concluído, Cancelado).
- Permitir classificação (crescente/decrescente) por qualquer coluna.
- Adicionar uma barra de pesquisa para filtrar pedidos por ID, nome do cliente ou data.

2. **Visualização de detalhes do pedido:**



- Criar uma janela ou seção separada para exibir os detalhes de um pedido selecionado.

- Exibir as seguintes informações:
    - ID do Pedido
    - Nome do Cliente
    - Data e Hora do Pedido
    - Valor Total
    - Status do Pedido
    - Lista de itens encomendados com quantidade e preço

3. **Atualizar status do pedido:**
    - Adicionar botões ou um menu suspenso para atualizar o status do pedido.
    - Status disponíveis: `"Processando"`, `"Concluído"`, `"Cancelado"`.
    - Usar a API para atualizar o status do pedido no banco de dados.

Entregáveis:

- Incorporar a interface do usuário (UI) ao arquivo Session3_DesktopApp.exe existente.

Notas adicionais:

- Usar sua estrutura de interface do usuário (UI) do .NET preferida.
- A interface do usuário (UI) deve ser responsiva e se adaptar a diferentes tamanhos de tela.
- O tratamento de erros e o feedback do usuário são essenciais (por exemplo, exibir mensagens quando um pedido é atualizado com sucesso ou se ocorre um erro).

3.5 Aceitação e Teste de Caixa Preta

Objetivo

Desenvolver testes de aceitação para verificar a funcionalidade da API e garantir que ela atenda aos requisitos especificados. Realizar testes de caixa preta para validar o comportamento da API sem conhecimento do funcionamento interno.

Tarefas

1. **Ambiente de Teste de Configuração**:
    - Certifique-se de que o aplicativo Flask esteja em execução localmente.
    - Usar o script test_backend.py fornecido para executar testes.
2. **Testes de Aceitação**:
    - **Endpoints do Produto**:
        - Verificar se GET /api/products retorna uma lista de todos os produtos.
        - Verificar se GET /api/products/{id} retorna os detalhes corretos do produto.
        - Verificar se o POST /api/products cria um novo produto com dados válidos.
        - Verificar se PUT /api/products/{id} atualiza um produto existente.
        - Verificar se DELETE /api/products/{id} exclui o produto especificado.

- **Endpoints do Pedido**:
    - Verificar se GET /api/orders retorna uma lista de todos os pedidos.
    - Verificar se GET /api/orders/{id} retorna os detalhes corretos do pedido.
    - Verificar se o POST /api/orders cria um novo pedido com dados válidos.
    - Verificar se PUT /api/orders/{id}/complete marca um pedido como concluído.
    - Verificar se PUT /api/orders/{id}/cancel cancela um pedido e restaura o estoque.

3. **Teste de Caixa Preta**:
    - Testar a API sem conhecimento da estrutura interna do código.
    - Concentre-se na validação de entrada-saída e no tratamento de erros.
    - Usar várias entradas de dados para testar casos extremos e cenários inválidos.

Entregáveis

- **Nome do Arquivo**: Session3_AcceptanceTests.zip
- **Tipo de Arquivo**: Arquivo compactado (.zip)
- **Conteúdo**:
    - Todos os arquivos de teste (arquivos .cs para .NET) para aceitação e solução de teste de caixa preta.
    - Todos os arquivos de configuração necessários.

Avisos Extras

- Usar técnicas de mocking ou stubbing para simular dependências externas.
- Certifique-se de que os testes sejam executados isoladamente e não dependam de recursos externos.
- Consultar o documento "Práticas Recomendadas de design de API Web" para obter diretrizes sobre como escrever testes eficazes.

3.6 Teste de Unidade e Caixa Branca

Objetivo:

O objetivo deste módulo é desenvolver e testar um sistema de gerenciamento de estoque de panificação usando técnicas de teste de unidade e caixa branca. Os concorrentes implementarão aulas para gerenciar itens de panificação e estoque, projetar casos de teste abrangentes e garantir a qualidade e a funcionalidade do código por meio de testes rigorosos.

Tarefas

1. **Implementar a classe BakeryItem**:
    - Criar uma classe chamada BakeryItem com propriedades para Name, Price, Quantity e ExpirationDate.
    - Implementar um método IsExpired() para determinar se o item expirou.
2. **Implementar a classe BakeryInventory**:

- Criar uma classe chamada BakeryInventory para gerenciar uma coleção de objetos BakeryItem.
- Implementar métodos para:
    - AddItem(BakeryItem item): adicionar um novo item ao inventário.
    - RemoveItems(nome da cadeia de caracteres): remova um item ou itens do inventário por nome.
    - GetTotalValue(): calcular o valor total (Preço * Quantidade) de todos os itens no inventário.

3. **Gravar Testes de Unidade**:
    - Usar nUnit para escrever testes de unidade para ambas as classes. Verificar o seguinte:
        - As propriedades BakeryItem são definidas e recuperadas corretamente.
        - O método IsExpired determina com precisão se um item expirou.
        - Os métodos BakeryInventory adicionam, removem e calculam corretamente o valor total dos itens.
        - Lidar com casos extremos, como tentar remover um item que não existe.
4. **Realizar Testes de Caixa Branca**:
    - Analisar o código para identificar caminhos críticos e possíveis áreas de risco.
    - Certifique-se de que todos os caminhos de execução possíveis sejam cobertos pelos casos de teste.

Entregáveis

- **Nome do Arquivo**: BakeryInventoryTesting.zip
- **Tipo de Arquivo**: Arquivo compactado (.zip)
- **Conteúdo**:
    - Arquivos de classe C# para BakeryItem e BakeryInventory.
    - Projetar do teste usando nUnit para testar ambas as classes.

Avisos Extras

- Verificar se o código está bem documentado com comentários explicando a lógica.
- Usar as práticas recomendadas para convenções de nomenclatura e organização de código.
- Certifique-se de que todos os testes unitários sejam aprovados antes do envio.