Checklist para o Projeto Final

1. Design e Implementação

O design do sistema está completamente e corretamente implementado?

Todas as funcionalidades descritas nas histórias de usuário foram implementadas?

Ex: Cadastro de usuários, busca de produtos, ordenação por preço/localização, etc.

As funções estão corretamente modularizadas e seguem o princípio de responsabilidade única?

Há funções ausentes ou desnecessárias no código?

O sistema permite a extensão para novas funcionalidades (ex: adicionar novos tipos de usuários ou produtos)?

2. Interfaces

O número de parâmetros das funções corresponde ao número de argumentos esperados?

Ex: cadastrar_usuario(nome, email, tipo) recebe exatamente 3 argumentos.

Os tipos e unidades dos parâmetros e argumentos estão corretos?

Ex: nome é uma string, preco é um float, etc.

As interfaces das funções estão consistentes com o design do sistema?

Ex: buscar_produto(produtos, nome) retorna uma lista de produtos que correspondem ao nome.

As variáveis globais são usadas corretamente e de forma consistente entre os módulos?

Ex: bancas e usuarios são listas globais que são manipuladas corretamente.

As constantes são usadas de forma apropriada?

Ex: O tipo padrão "cliente" em cadastrar_usuario é uma constante.

3. Dados e Armazenamento

Todas as variáveis são explicitamente declaradas e inicializadas antes do uso?

Ex: bancas = [] e usuarios = [] são inicializados corretamente.

Os dados são armazenados e acessados corretamente?

Ex: Produtos são armazenados em uma lista de dicionários com campos como nome, preco, banca, etc.

As estruturas de dados (listas, dicionários) são usadas de forma eficiente?

Ex: buscar_produto usa uma lista de dicionários para buscar produtos.

Os índices de arrays e strings estão dentro dos limites?

Ex: Não há acesso a índices inválidos em listas ou strings.

Os ponteiros (se aplicável) são usados corretamente?

Ex: Não há vazamentos de memória ou uso incorreto de referências.

4. Manutenibilidade e Testabilidade

O código é fácil de entender?

Ex: Nomes de variáveis e funções são descritivos (cadastrar_usuario, buscar_produto).

Há comentários suficientes para explicar o código?

Ex: Funções como cadastrar_banca têm comentários explicando seu propósito.

O código segue um padrão de formatação e indentação consistente?

Ex: O código usa indentação de 4 espaços e segue o estilo PEP 8 (para Python).

O código está estruturado para facilitar testes?

Ex: As funções são testáveis individualmente (ex: test_cadastrar_usuario).

O código segue as convenções de codificação definidas?

Ex: Uso de snake_case para nomes de funções e variáveis em Python.

O código evita lógica complexa ou obscura?

Ex: A lógica de busca e ordenação é clara e direta.

5. Tratamento de Erros

Todas as condições de erro possíveis são tratadas?

Ex: O que acontece se um produto não for encontrado em buscar_produto?

Mensagens de erro são claras e úteis?

Ex: "Produto não encontrado!" é uma mensagem clara para o usuário.

O código faz verificações de limites e valores inválidos?

Ex: Verifica se o preço de um produto é um número positivo.

O código permite a recuperação de erros?

Ex: Se uma banca não for encontrada, o sistema continua funcionando.

Os casos padrão em estruturas condicionais são tratados corretamente?

Ex: O que acontece se nenhum critério de ordenação for especificado em ordenar_produtos?

6. Testes

O código foi desenvolvido de forma orientada a testes?

Ex: As funções têm testes unitários correspondentes (ex: test_cadastrar_usuario).

Os testes cobrem todos os cenários possíveis?

Ex: Testes para cadastro, busca, ordenação e tratamento de erros.

O framework de testes escolhido foi utilizado corretamente?

Ex: Uso de unittest ou pytest para Python.

A cobertura de testes é suficiente?

Ex: Ferramentas como coverage.py são usadas para verificar a cobertura.

7. Documentação

O código está documentado de acordo com as ferramentas escolhidas?

Ex: Uso de Doxygen ou Sphinx para gerar documentação.

A documentação inclui exemplos de uso das funções?

Ex: Como usar cadastrar_banca ou buscar_produto.

A documentação está atualizada com as últimas mudanças no código?

8. Metodologias e Ferramentas

O projeto segue as metodologias Scrum, Kanban e XP?

Ex: Uso de ferramentas como Trello para gerenciamento de tarefas.

O controle de versão está sendo utilizado corretamente?

Ex: Uso de Git e GitHub para versionamento do código.

As bibliotecas e frameworks utilizados são de código aberto e possuem licenças compatíveis?

Ex: Uso de bibliotecas como pytest (MIT License) ou Flask (BSD License).

9. Relatórios e Métricas

O sistema gera relatórios para o administrador?

Ex: Quantidade de usuários, produtos mais pesquisados, etc.

As métricas de uso do sistema estão sendo coletadas e analisadas?

Ex: Número de buscas realizadas, produtos mais vendidos, etc.