Universidade Tuiuti do Paraná
Curso: Ciência da Computação
2º Estudo Dirigido de Programação Avançada - 12/05/25
Prof. Baroni

Estudo Dirigido: Projeto SoundWave - Player de Músicas

Em um cenário onde as grandes corporações dominam o mercado de streaming musical, um grupo de estudantes de tecnologia decide criar uma alternativa independente e inovadora. Nasce assim o projeto *SoundWave*, um player de músicas desenvolvido com Python que promete combinar a simplicidade dos players tradicionais com recursos avançados normalmente encontrados apenas em aplicações comerciais de grande porte. A equipe da SoundWave acredita que a música deve ser acessível e controlada pelos próprios ouvintes, não por algoritmos opacos que decidem o que você deve escutar. Para isso, vocês foram contratados como desenvolvedores para criar o protótipo funcional do player, respeitando a filosofia de código aberto e enfrentando desafios técnicos empolgantes.

O CEO da startup, Alex Rivera, compartilhou a visão: "Queremos criar uma experiência musical que dê controle total ao usuário. Playlists inteligentes, gerenciamento eficiente de filas, histórico completo... e tudo isso funcionando de forma fluida, mesmo quando o usuário está importando novas músicas ou gerando estatísticas de uso. O SoundWave será o player para os verdadeiros amantes da música e da tecnologia."

Objetivo do Projeto
Desenvolver um player de músicas em Python que implemente conceitos avançados de programação, demonstrando o domínio de estruturas de dados, padrões de projeto, multithreading e boas práticas de desenvolvimento.

Objetivo do Projeto:

Desenvolver um player de músicas funcional em Python que demonstre o uso adequado de:

- Princípios de Orientação a Objetos
- Estruturas de dados avançadas
- Técnicas de manipulação de arquivos
- Serialização de objetos
- Programação concorrente
- Padrões de projeto relevantes
- Modularização eficiente
- Práticas de teste adequadas
- Requisitos Funcionais

Requisitos Funcionais:

- Gerenciamento de Biblioteca Musical
    - Permitir a adição de arquivos de música à biblioteca
    - Organizar músicas por artista, álbum, gênero, etc.
    - Permitir buscas e filtros na coleção de músicas
- Reprodução de Áudio
    - Reproduzir arquivos de áudio em formatos comuns (mp3, wav, etc.)
    - Controles básicos: play, pause, stop, próxima, anterior
    - Controle de volume
- Gerenciamento de Playlists

    - Criar, editar e excluir playlists
    - Adicionar e remover músicas das playlists
    - Ordenar músicas dentro das playlists
- Persistência de Dados
    - Salvar e carregar o estado da biblioteca e playlists
    - Manter histórico de reprodução
    - Armazenar configurações do usuário
- Funcionalidades Adicionais Extras (pontos bônus)
    - Sistema de recomendações simples
    - Visualização de estatísticas de reprodução
    - Equalização básica de áudio
    - Suporte a favoritos/marcações
    - Downloads e gestão de cache local

Requisitos Técnicos:

1. Orientação a Objetos
    a. Utilizar herança, polimorfismo, encapsulamento
    b. Implementar classes coesas e com responsabilidades bem definidas
    c. Aplicar princípios SOLID onde apropriado

2. Estruturas de Dados
    a. O sistema deve utilizar pelo menos três das seguintes estruturas:
    b. Listas encadeadas para sequência de reprodução
    c. Filas para gerenciamento de reprodução
    d. Pilhas para histórico de músicas reproduzidas
    e. Árvores para organização hierárquica da biblioteca
    f. Tabelas hash para indexação e busca rápida
    g. Grafos para sistemas de recomendação

3. Manipulação de Arquivos e Serialização
    a. Leitura de metadados de arquivos de áudio
    b. Serialização e desserialização de objetos para persistência
    c. Leitura e escrita eficiente de dados em disco

4. Programação Concorrente
    a. Utilizar multithreading para operações simultâneas
    b. Implementar processamento paralelo onde apropriado
    c. Garantir sincronização adequada entre threads

5. Padrões de Projeto - Implementar pelo menos três dos seguintes padrões:
    a. Singleton para gerenciadores de recursos
    b. Observer para notificações de eventos do player
    c. Factory/Builder para criação de objetos complexos
    d. Adapter para integração com bibliotecas de áudio
    e. Command para operações do player
    f. Strategy para diferentes comportamentos de reprodução
    g. Decorator para funcionalidades adicionais em tempo de execução
    h. Repository para acesso a dados

6. Modularização e Organização
    a. Criar módulos Python bem organizados
    b. Definir interfaces claras entre componentes
    c. Separar responsabilidades em pacotes coerentes

7. Testes
    a. Implementar testes unitários para componentes críticos
    b. Realizar testes de integração para fluxos principais

Entregáveis

1. Código-fonte completo
2. Organizado, comentado e seguindo PEP 8
3. Requisitos em arquivo requirements.txt
4. README com instruções de instalação e execução
5. Relatório Técnico (PDF, 10-15 páginas)

Descrição da arquitetura do sistema

1. Justificativas para decisões técnicas importantes
2. Explicação dos padrões de projeto utilizados
3. Diagrama de classes UML
4. Desafios encontrados e soluções implementadas
5. Análise crítica do resultado final
6. Possíveis melhorias futuras

Critérios de Avaliação

Critérios	Peso	Explicação
Funcionalidade	30%	O programa funciona corretamente com todos os requisitos funcionais?
Arquitetura	20%	O design é bem estruturado, modular e segue princípios de OO?
Implementação	15%	Uso adequado de estruturas de dados, padrões de design e multithreading;
Uso correto de Padrões de Projeto;	15%	Presença de pelo menos três padrões diferentes;
Relatório técnico;	10%	Completo, bem formatado, e deixando claro as contribuições de cada um;
Testes;	10%	Escrever testes para pelo menos 70% do código;
Criatividade (Bônus);	15%	

Instruções para Entrega

- Data limite: 27 de junho de 2025, às 23:59
- Formato: arquivos fonte + documentação em PDF
- Via TEAMS da Disciplina
- Trabalho pode ser feito em trios;

Dicas e Recursos

1. Comece simples e evolua
2. Implemente primeiro um player básico funcional
3. Adicione recursos gradualmente
4. Planeje antes de codificar
5. Um bom diagrama UML economizará tempo depois
6. Defina interfaces claras entre componentes
7. Foque na qualidade das implementações em vez da quantidade
8. Documente limitações conhecidas e possíveis expansões futuras

Recursos de aprendizado

9. Design Patterns em Python: https://refactoring.guru/design-patterns/python
10. Multithreading em Python: https://realpython.com/intro-to-python-threading/
11. Audio em Python: https://realpython.com/playing-and-recording-sound-python/

Perguntas
P: Posso usar bibliotecas como spotify-api ou similar?
R: Não para funcionalidades principais. O objetivo é implementar você mesmo os mecanismos core do player.

P: É necessário suportar streaming de áudio da internet?
R: Não é obrigatório, mas pode ser implementado como recurso adicional.

P: Como devo lidar com arquivos de áudio para testes?
R: Utilize arquivos de domínio público ou crie arquivos de áudio para teste.

P: A interface gráfica é obrigatória?
R: Não, mas oferece pontos bônus. Uma interface de linha de comando bem feita também é aceitável.

Boa sorte e bom desenvolvimento!