Documentação - Módulo Admin

Índice

Visão Geral

O módulo **admin.cpp** é responsável por todas as funcionalidades administrativas do sistema de gestão da Escola de Música Benicasa. Fornece ao administrador ferramentas para:

- **Gestão de Usuários**: Cadastro, ativação e inativação de alunos, professores e administradores
- **Gestão Acadêmica**: Cursos, disciplinas, turmas e matrículas
- **Gestão de Eventos**: Criação, autorização e gerenciamento de eventos
- **Gestão de Instrumentos**: Cadastro, autorização e controle de estoque
- **Gestão de Produtos**: Lanchonete e gerenciamento de inventário
- **Relatórios**: Financeiro, patrimonial e acadêmico
- **Backup e Restauração**: Sistema de backup automático com timestamp

---

Estrutura do Módulo

```
admin.cpp/
├── Funções Auxiliares Gerais
│   ├── limparbuffer()
│   └── openFile()
├── Funções de Busca
│   ├── buscaProf()
│   ├── buscaDisciplina()
│   ├── buscaAluno()
│   ├── buscarInstrumento()
│   ├── buscaProduto()
│   └── gerarNovoId()
├── Funções de Consulta e Relatórios
│   └── consultarPendenciasInstrumentos()
└── Namespace mod_ADM
    ├── Funções Auxiliares de Listagem
    ├── Menus Principais
    ├── Funções de Turmas e Matrícula
    ├── Menu de Produtos
```

```
    ├── Menu de Instrumentos
    └── Relatórios e Backup
```

Funções Auxiliares

`void limparbuffer()`

Localização: Linha 20-22

Limpa o buffer de entrada para evitar problemas de leitura de dados.

Uso: Chamada antes de operações de entrada de texto para garantir limpeza do buffer.

`void openFile(std::fstream &f, const std::string Nome)`

Localização: Linha 24-32

Abre um arquivo em modo binário para leitura e escrita.

Parâmetros:

- `fstream &f`: Referência ao arquivo a ser aberto
- `string Nome`: Nome do arquivo (sem extensão .dat)

Modo de operação: Abre em modo `ios::in | ios::out | ios::binary`

Exemplo:

```
fstream file;
openFile(file, "alunos");
// Agora file está aberto e pronto para uso
file.close();
```

Funções de Busca

`Professor buscaProf(std::fstream &file, int buscaId)`

Localização: Linha 39-60

Busca um professor pelo ID no arquivo de professores.

Parâmetros:

- `fstream &file`: Arquivo aberto de professores
- `int buscaId`: ID do professor a buscar

Retorno: Estrutura `Professor` (retorna struct vazio se não encontrado)

Comportamento: Busca sequencial pelo ID

`Disciplina buscaDisciplina(std::fstream &file, int buscaId)`

Localização: Linha 63-83

Busca uma disciplina pelo ID.

Parâmetros:

- `fstream &file`: Arquivo aberto de disciplinas
- `int buscaId`: ID da disciplina

Retorno: Estrutura `Disciplina`

`Aluno buscaAluno(std::fstream &file, int buscaId)`

Localização: Linha 86-107

Busca um aluno pelo ID.

Parâmetros:

- `fstream &file`: Arquivo aberto de alunos
- `int buscaId`: ID do aluno

Retorno: Estrutura `Aluno`

`Instrumento buscarInstrumento(std::fstream &file, int buscaId)`

Localização: Linha 110-130

Busca um instrumento pelo ID.

Parâmetros:

- `fstream &file`: Arquivo aberto de instrumentos
- `int buscaId`: ID do instrumento

Retorno: Estrutura `Instrumento`

`Produto buscaProduto(std::fstream &file, int buscaId)`

Localização: Linha 133-156

Busca um produto (lanchonete) pelo ID.

Parâmetros:

- `fstream &file`: Arquivo aberto de produtos
- `int buscaId`: ID do produto

Retorno: Estrutura `Produto`

`int gerarNovoId(std::fstream &file, size_t tamanhoStruct)`

Localização: Linha 159-164

Gera um novo ID sequencial para novos registros.

Parâmetros:

- `fstream &file`: Arquivo aberto
- `size_t tamanhoStruct`: Tamanho da estrutura de dados

Retorno: Novo ID (baseado na quantidade de registros)

Lógica: ID = 1 + (tamanho_do_arquivo / tamanho_da_estrutura)

Funções de Consulta e Relatórios

`void consultarPendenciasInstrumentos()`

Localização: Linha 169-337

Menu para consultar e gerenciar instrumentos com pendências.

Operações:

1. Visualizar instrumentos pendentes
2. Gerenciar pendências
3. Voltar

Fluxo:

- Exibe tabela de instrumentos com status de autorização
- Permite marcar instrumentos como consultados
- Oferece opções de gerenciamento

Namespace mod_ADM

O namespace `mod_ADM` encapsula todas as funções de administração organizadas em seções:

Seção 1: Funções Auxiliares de Listagem

`void atualizar_estado_de_usuario(int id_usuario, Funcao tipo_usuario, string estado)`

Localização: Linha 352-393

Atualiza o estado de ativação de um usuário.

Parâmetros:

- `int id_usuario`: ID do usuário
- `Funcao tipo_usuario`: Tipo (ALUNO, PROFESSOR, ADMINISTRADOR)
- `string estado`: Novo estado

Tipos suportados: Trabalha com arquivos específicos por tipo de usuário

---

`int listar_usuarios_especificos(Funcao tipo_usuario, int ativo, string dados[100][6])`

Localização: Linha 397-457

Lista usuários filtrados por tipo e status de ativação.

Parâmetros:

- `Funcao tipo_usuario`: ALUNO, PROFESSOR ou ADMINISTRADOR
- `int ativo`: Filtro (1=Ativo, 0=Inativo, 2=Ambos)
- `string dados[100][6]`: Array para armazenar os dados

Retorno: Quantidade de usuários encontrados

Colunas de saída:

-
-
-
-
-
-

`int listar_disciplinas_especificas(int ativo, string dados[100][6])`

Localização: Linha 460-475

Lista disciplinas com filtro de ativação.

Parâmetros:

- `int ativo`: 0=Inativas, 1=Ativas, 2=Ambas
- `string dados[100][6]`: Array para armazenar dados

Retorno: Quantidade de disciplinas

`int listar_eventos_especificos(int autorizado, string dados[100][5])`

Localização: Linha 478-496

Lista eventos com filtro de autorização.

Parâmetros:

- `int autorizado`: 0=Não autorizados, 1=Autorizados, 2=Ambos
- `string dados[100][5]`: Array para armazenar dados

Retorno: Quantidade de eventos

Colunas:

-
-
-
-
-

`int listar_instrumentos_especificos(int autorizado, string dados[100][6])`

Localização: Linha 501-518

Lista instrumentos com filtro de autorização.

Parâmetros:

- `int autorizado`: 0=Não autorizados, 1=Autorizados, 2=Ambos
- `string dados[100][6]`: Array para armazenar dados

Retorno: Quantidade de instrumentos

Seção 2: Menus Principais

`void menuCadastroUsuarios()`

Localização: Linha 530-570

Menu para cadastro de alunos, professores e administradores.

Opções:

1. Cadastrar Aluno
2. Cadastrar Professor
3. Voltar

`void menuGerenciarUsuarios()`

Localização: Linha 572-750

Menu completo para gerenciamento de usuários existentes.

Funcionalidades:

- Filtrar por tipo (Aluno, Professor, Administrador)
- Filtrar por status (Ativo, Inativo, Ambos)

- Listar usuários em tabela interativa
- Editar usuário selecionado
- Buscar usuário por ID
- Ativar/Inativar usuários

Fluxo:

1. Seleciona tipo de usuário
2. Seleciona filtro de status
3. Exibe tabela com usuários
4. Permite edição ou busca

Seção 3: Menu de Cursos/Disciplinas

`void menuCadastroCursos()`

Localização: Linha 755-1023

Menu principal para gerenciamento de disciplinas.

Opções Disponíveis:

#	Opção	Função
0	Cadastrar Disciplina	Cria nova disciplina com nome e carga horária
1	Autorizar Disciplina	Muda status de inativa para ativa
2	Inativar Disciplina	Desativa disciplina ativa
3	Vincular Professor	Associa professor à disciplina
4	Listar Disciplinas	Exibe todas as disciplinas em tabela
5	Voltar	Retorna ao menu anterior

Fluxo de Cadastro:

```
Entrada: Nome da Disciplina + Carga Horária
   ↓
Validação
   ↓
Gera novo ID
   ↓
Salva com status inativo (aguarda autorização)
   ↓
Confirmação de sucesso
```

Fluxo de Autorização:

```
Listar disciplinas inativas
   ↓
Selecionar disciplina em tabela
   ↓
Confirmar autorização
   ↓
Mudar status para ativo
   ↓
Registrar professor responsável
```

`void consultarRelatoriosAcademicos()`

Localização: Linha 1026-1063

Relatório de desempenho acadêmico dos alunos.

Dados Exibidos:

- ID do aluno
- Nome completo
- Email
- Total de faltas
- Status (Ativo/Inativo)

Características:

- Exclui aluno padrão (ID 20260000)
- Mostra apenas alunos ativos
- Tabela interativa

Seção 4: Menu de Eventos

`void menuEventos(std::fstream &file)`

Localização: Linha 1068-1327

Gerenciamento completo de eventos.

Opções:

#	Opção	Descrição
0	Cadastrar Evento	Criar novo evento com detalhes
1	Autorizar Evento	Aprovar eventos pendentes
2	Inativar Evento	Desativar eventos ativos
3	Listar Eventos	Visualizar eventos em tabela

#	Opção	Descrição
4	Voltar	Retornar

Campos de Cadastro:

- Nome do evento
- Local
- Descrição
- Data (formato dd/mm/aaaa)
- Total de vagas
- Status inicial: Não autorizado
- Vagas ocupadas: 0

Estados de um Evento:

- `ativo`: 1 (ativo) ou 0 (inativo)
- `autorizado`: 1 (autorizado) ou 0 (pendente)

Seção 5: Funções de Turmas e Matrícula

`bool verificaTurmasProf(Professor &prof, int &Index_turma)`

Localização: Linha 1330-1336

Verifica se professor tem vaga para nova turma.

Parâmetros:

- `Professor &prof`: Referência ao professor
- `int &Index_turma`: Índice da vaga encontrada

Retorno: `true` se houver vaga, `false` caso contrário

Limite: Máximo 5 turmas por professor

`void cadastrarTurma()`

Localização: Linha 1338-1425

Cadastra uma nova turma ligando disciplina e professor.

Processo:

1. Solicita ID da disciplina
2. Valida se disciplina existe e está ativa
3. Solicita ID do professor
4. Valida se professor existe
5. Verifica vaga no professor
6. Atualiza turmas do professor

7. Cria turma com capacidade máxima

Validações:

- Disciplina deve existir
- Professor deve existir
- Professor não pode ter 5+ turmas
- Disciplina deve estar autorizada

`void matricularAlunoTurma()`

Localização: Linha 1427-1517

Matricula um aluno em uma turma específica.

Processo:

1. Solicita ID do aluno
2. Valida existência do aluno
3. Solicita ID da turma
4. Valida existência e disponibilidade da turma
5. Verifica se aluno já está na turma
6. Localiza vaga na turma
7. Adiciona aluno e incrementa contagem

Validações:

- Aluno deve estar ativo
- Turma deve estar ativa
- Turma não deve estar cheia
- Aluno não pode estar duplicado na turma

Seção 6: Menu de Instrumentos

`void menuCadastroInstrumentos()`

Localização: Linha 1519-1896

Menu completo para gerenciamento de instrumentos.

Opções:

#	Opção	Função
0	Cadastrar Instrumento	Registra novo instrumento
1	Autorizar Instrumento	Aprova instrumentos pendentes
2	Inativar Instrumento	Desativa instrumentos
3	Listar Instrumentos	Exibe todos em tabela

#	Opção	Função
4	Voltar	Retornar

Campos de Cadastro:

- Nome do instrumento
- Quantidade em estoque
- Status inicial: Ativo (1), Não autorizado (0)

Status de um Instrumento:

- `ativo`: Instrumento existe no sistema
- `autorizado`: Visibilidade/disponibilidade
- `disponivel`: Livre para empréstimo
- `estoque`: Quantidade disponível

Seção 7: Menu de Produtos (Lanchonete)

`void cadastrarProdutos()`

Localização: Linha 1898-1900

Delega para menu de produtos da lanchonete.

```
// Chama diretamente a função da classe Lanchonete
Lanchonete::menuCadastroProdutos();
```

Seção 8: Relatórios e Backup

`void gerarRelatorioFinanceiro()`

Localização: Linha 1903-1942

Gera relatório financeiro da lanchonete.

Dados:

- ID do produto
- Nome
- Quantidade em estoque
- Preço unitário
- Total (quantidade × preço)

Resumo:

- Total de operações
- Total de entradas (R$)

Nota: Calcula valor total em estoque

`void gerarRelatorioPatrimonial()`

Localização: Linha 1943-2018

Relatório de bens e instrumentos da instituição.

Dados por Instrumento:

- ID
- Nome
- Estoque
- Status (Disponível/Emprestado)
- Emprestado para (nome do aluno)
- Data prevista de devolução

Resumo:

- Total de instrumentos
- Disponíveis
- Emprestados

Relacionamentos:

- Cruza dados de `instrumentos.dat` e `emprestimos.dat`

`void realizarBackup()`

Localização: Linha 2020-2063

Realiza backup automático com timestamp.

Características:

- **Pasta**: `backup/YYYY-MM-DD_HH-MM-SS/`
- **Arquivos .dat**:
  - alunos.dat
  - professores.dat
  - administradores.dat
  - disciplinas.dat
  - eventos.dat
  - instrumentos.dat
  - emprestimos.dat
  - usuarios.dat
  - notas.dat
- **Arquivos .txt**:

    - cadastros.txt
    - lanchonete.txt
    - instrumentos.txt
    - eventos.txt

Processo:

1. Gera timestamp
2. Cria pasta com data/hora
3. Copia arquivos .dat e .txt
4. Conta arquivos copiados
5. Exibe confirmação

`void restaurarBackup()`

Localização: Linha 2065-2171

Restaura dados de um backup anterior.

Processo:

1. Lista backups disponíveis
2. Exibe em tabela interativa
3. Oferece opções:
    - **Restaurar**: Sobrescreve arquivos atuais
    - **Apagar**: Remove backup
    - **Voltar**: Cancela

Confirmação:

- Mostra data/hora do backup
- Conta arquivos restaurados

Cuidado: Sobrescreve dados atuais sem confirmação adicional

Menu Principal

`void abrir_menu_admin(Usuario* usuario)`

Localização: Linha 2041-2115

Ponto de entrada principal do módulo administrativo.

Parâmetro:

- `Usuario* usuario`: Ponteiro para usuário logado

Opções do Menu:

#	Opção	Função Chamada

#	Opção	Função Chamada
1	Cadastrar Cursos	`mod_ADM::menuCadastroCursos()`
2	Lanchonete - Consultar Estoque	`Lanchonete::consultarEstoque()`
3	Consultar Pendências	`consultarPendenciasInstrumentos()`
4	Gerenciar Eventos	`mod_ADM::menuEventos()`
5	Gerenciar Instrumentos	`mod_ADM::menuCadastroInstrumentos()`
6	Lanchonete - Gerenciar Produtos	`mod_ADM::cadastrarProdutos()`
7	Gerenciar Usuários	`mod_ADM::menuGerenciarUsuarios()`
8	Realizar Backup	`mod_ADM::realizarBackup()`
9	Relatório Financeiro	`mod_ADM::gerarRelatorioFinanceiro()`
10	Relatório Patrimonial	`mod_ADM::gerarRelatorioPatrimonial()`
11	Relatórios Acadêmicos	`mod_ADM::consultarRelatoriosAcademicos()`
12	Restaurar Backup	`mod_ADM::restaurarBackup()`
13	Logout	`usuario->logado = false`

Comportamento:

- Menu baseado em seleção por índice
- Exibe nome de boas-vindas do administrador
- Usa interface gráfica para apresentação

Fluxo de Dados

Fluxo Geral de Leitura

```
openFile(file, "nome.dat")
   ↓
file.seekg(posição)
   ↓
file.read((char*)&struct, sizeof(struct))
   ↓
// Processar dados
   ↓
file.close()
```

Fluxo Geral de Escrita

```
openFile(file, "nome.dat")
   ↓
```

```
file.seekp(posição)
   ↓
file.write((char*)&struct, sizeof(struct))
   ↓
file.clear()
   ↓
file.close()
```

Fluxo de Geração de ID

```
gerarNovoId(file, sizeof(Struct))
   ↓
Calcula: 1 + (tellp() / sizeof(Struct))
   ↓
Retorna: Novo ID sequencial
```

Fluxo de Interface Tabular

```
Coletar dados em array 2D
   ↓
Preparar ponteiros de strings
   ↓
Configurar interface_para_tabela()
   ↓
Obter selação do usuário
   ↓
Processar seleção
```

Estruturas de Dados Relacionadas

Estrutura de Usuário Base

```
struct Usuario {
    int id;                 // ID único
    char nome[50];          // Nome completo
    char email[50];         // Email
    char senha_hash[100];   // Senha criptografada
    Funcao categoria;       // ALUNO, PROFESSOR, ADMINISTRADOR
    int ativo;              // 1 = Ativo, 0 = Inativo
    bool logado;            // Status de login
};
```

Estrutura de Disciplina

```
struct Disciplina {
    int id;                 // ID único
    char nome[30];          // Nome da disciplina
    int cargaHoraria;       // Horas de aula
    int ativo;              // 1 = Ativo, 0 = Inativo
};
```

Estrutura de Evento

```
struct Evento {
    int id;                 // ID único
    char nome[50];          // Nome do evento
    char local[50];         // Local
    char descricao[100];    // Descrição
    char data[11];          // dd/mm/aaaa
    int totalVagas;         // Capacidade
    int vagasOcupadas;      // Vagas utilizadas
    int autorizado;         // 1 = Autorizado, 0 = Pendente
    int ativo;              // 1 = Ativo, 0 = Inativo
};
```

Estrutura de Instrumento

```
struct Instrumento {
    int id;                 // ID único
    char nome[30];          // Nome
    int estoque;            // Quantidade
    int autorizado;         // 1 = Autorizado, 0 = Pendente
    int ativo;              // 1 = Ativo, 0 = Inativo
    int disponivel;         // 1 = Disponível, 0 = Emprestado
};
```

Convenções Utilizadas

Nomes de Arquivos

- Arquivos de dados: `*.dat` (binários)
- Arquivos de texto auxiliar: `*.txt`
- Backups: `backup/YYYY-MM-DD_HH-MM-SS/`

Valores de Status

Campo	Ativo	Inativo
ativo	1	0

Campo	Ativo	Inativo
autorizado	1	0
logado	true	false

Identificadores Especiais

- ID padrão do aluno de teste: `20260000`
- Máximo de turmas por professor: `5`
- Máximo de alunos por turma: `MAX_ALUNOS` (definido em headers.h)

Notas Importantes

1. **Segurança de Arquivo**: Sempre feche arquivos com `file.close()` após uso
2. **Buffer de Entrada**: Chame `limparbuffer()` antes de operações IO críticas
3. **Limite de Tabelas**: Arrays de dados limitados a 100 registros
4. **Posicionamento de Arquivo**: Use `seekg()` (leitura) e `seekp()` (escrita) corretamente
5. **Cálculo de Posição**: `(ID - 1) * sizeof(Struct)` ou `(ID - offset) * sizeof(Struct)`
6. **Backup**: Realizado com sucesso independente do sistema (Windows/Linux)
7. **Relatórios**: Mostra dados diretamente sem exportação para arquivo

---

Diagrama de Dependências

```
abrir_menu_admin()
├── menuCadastroCursos()
│   ├── buscaDisciplina()
│   ├── buscaProf()
│   └── listar_disciplinas_especificas()
├── menuGerenciarUsuarios()
│   ├── listar_usuarios_especificos()
│   └── atualizar_estado_de_usuario()
├── menuEventos()
│   └── listar_eventos_especificos()
├── menuCadastroInstrumentos()
│   └── listar_instrumentos_especificos()
├── cadastrarTurma()
│   ├── buscaDisciplina()
│   ├── buscaProf()
│   └── verificaTurmasProf()
├── matricularAlunoTurma()
│   ├── buscaAluno()
│   └── [validações]
├── gerarRelatorioFinanceiro()
├── gerarRelatorioPatrimonial()
│   └── [leitura de múltiplos arquivos]
├── realizarBackup()
│   └── [cópia de arquivos]
```

```
└── restaurarBackup()
    └── [restauração de arquivos]
```

Exemplo de Uso Completo

Cadastro de Disciplina

```
// 1. Abrir menu do admin
Usuario admin;
admin.nome = "João";
abrir_menu_admin(&admin);

// 2. Usuário seleciona opção 1 (Cadastrar Cursos)
// 3. Sistema abre menuCadastroCursos()

// 4. Usuário seleciona opção 0 (Cadastrar Disciplina)
// 5. Sistema solicita:
//    - Nome: "Violino Básico"
//    - Carga Horária: 60

// 6. Sistema:
//    - Abre arquivo "disciplinas.dat"
//    - Gera novo ID
//    - Cria struct Disciplina
//    - Define status = inativo (0)
//    - Escreve no arquivo
//    - Exibe confirmação
```

Contato e Manutenção

Para dúvidas ou melhorias do módulo admin, consulte a documentação de interfaces gráficas em DOCUMENTACAO_INTERFACE_GRAFICA.md.