

Dendê Eventos (Descobrindo Kotlin)

Autores: Eli K. C. Barreto
Elias G. C. Barreto
Emerson L. S. Lima

Docente: Lucas Almeida Silva
Disciplina: Desenvolvimento Mobile Nativo

Agenda

O objetivo da apresentação é, para cada User Story do produto, demonstrar a aplicação dos tópicos seguintes.

Tópico 1: Apresentação da funcionalidade

Contextualizar a User Story, destacando objetivos.

Tópico 2: Sua estrutura de dados

Explicar data class/enum criado(s) e estrutura(s) utilizadas (listas/mapas/conjuntos) para a funcionalidade.

Tópico 3: Experiência do Usuário

Apresentar imagens de programa executando no terminal com objetivo de simular o uso popular.

Tópico 4: Validações e regras de negócio

Explicar condições impostas a saída de um loop, por exemplo.

Tópico 5: Apresentação da implementação da funcionalidade

Demonstrar como está implementada no geral uma funcionalidade.

User Story 3: Alterar Perfil do Usuário

- Objetivo

Um usuário do serviço deve ter acesso a alteração dos próprios dados cadastrais. Portanto, deve conseguir realizar alterações em seu nome, sua data de nascimento, sexo, senha, CNPJ, razão social e nome fantasia.



Estrutura de dados: Alterar Perfil do Usuário

- Data class

Para armazenar (em memória) dados de usuários, é usado um data class (Usuário).

```
data class Usuario(  
    var statusConta: Boolean,  
    var nome: String,  
    var dataNascimento: String,  
    var sexo: SexoUsuario,  
    val email: String,  
    var senha: String,  
    val tipoUsuario: TipoUsuario,  
    var cnpj: String? = null,  
    var razaoSocial: String? = null,  
    var nomeFantasia: String? = null  
)
```



Estrutura de dados: Alterar Perfil do Usuário

- Enum class

Para dados especificados/predefinidos, são usados enum classes (SexoUsuario, TipoUsuario).

```
enum class SexoUsuario { MASCULINO, FEMININO, NAO_INFORMADO }
enum class TipoUsuario { COMUM, ORGANIZADOR }
```

Estrutura de dados: Alterar Perfil do Usuário

- List

Para atualizar o data class (Usuario), é usada uma lista mutável (listaUsuarios).

```
val listaUsuarios = mutableListOf<Usuario>()
```



Experiência do Usuário: Alterar Perfil Usuário

MENU: Alterando EMERSON L S LIMA (241030866@aluno.unex.edu.br).

OPÇÕES:

[0] Voltar

[1] Nome [2] Data de Nascimento [3] Sexo [4] Senha

[5] CNPJ [6] Razão Social [7] Nome Fantasia

Digite opção:

4

OK: Digite Senha atualizada: *OrganizadorDeEventos*

Confirme Senha atualizada: *OrganizadorDeEventos*

OK: SENHA ATUALIZADA.

MENU: Alterando EMERSON L S LIMA (241030866@aluno.unex.edu.br).

OPÇÕES:

[0] Voltar

Regras de negócio, validações: Alterar Perfil

- E-mail cadastrado fixo (não é uma opção no menu de alteração)
- Usuários comuns não acessam opções de usuários organizadores

```
println("MENU: Alterando ${usuarioEncontrado.nome} (${usuarioEncontrado.email}).")  
println("OPÇÕES:")  
println("[0] Voltar\n[1] Nome [2] Data de Nascimento [3] Sexo [4] Senha")  
when {  
    usuarioEncontrado.tipoUsuario == TipoUsuario.ORGANIZADOR -> println(  
        "[5] CNPJ [6] Razão Social [7] Nome Fantasia"  
    )  
}  
println("Digite opção: ")
```

Regras de negócio, validações: Alterar Perfil

- Data de nascimento válida (dia 1..31; mês 1..12; ano 1920..2020)

```
dataValida = false
when {
    diaNascimento in 1 .. ≤ 31 && mesNascimento in 1 .. ≤ 12 && anoNascimento in 1920 .. ≤ 2020 -> {
        dataValida = true
        usuarioEncontrado.dataNascimento = "$diaNascimento/$mesNascimento/$anoNascimento"
    }
}

else -> println("ERRO: Data inválida. Tente novamente.")
}
```

Regras de negócio, validações: Alterar Perfil

- Senha deve ter ao menos oito dígitos e ser digitada duas vezes

```
var cadastroSenha: String
do {
    print("OK: Digite Senha atualizada: ")
    cadastroSenha = readln()
    print("Confirme Senha atualizada: ")
    val confirmarSenha = readln()
    when {
        cadastroSenha.length < 8 -> println("ERRO: A senha deve ter no mínimo 8 caracteres.\n")
        cadastroSenha != confirmarSenha -> println("ERRO: Senhas não conferem. Tente novamente.")
        else -> println("OK: SENHA ATUALIZADA.\n")
    }
} while (cadastroSenha.length < 8 || cadastroSenha != confirmarSenha)
usuarioEncontrado.senha = cadastroSenha
```

Regras de negócio, validações: Alterar Perfil

- CNPJ somente em caso de usuário organizador e com 14 dígitos

```
when {  
    usuarioEncontrado.tipoUsuario == TipoUsuario.ORGANIZADOR -> {  
        var cnpjValido = false  
        var alterarCNPJ: String  
        do {  
            print("Digite CNPJ atualizado (14 dígitos): ")  
            alterarCNPJ = readln()  
  
            when {  
                alterarCNPJ.length == 14 -> cnpjValido = true  
                else -> println("ERRO: CNPJ inválido. Tente novamente.\n")  
            }  
        } while (!cnpjValido)
```

Implementação: Alterar Perfil do Usuário

```
var menuAlterarUsuario = true

// Alterar dados de conta ativa
do {
    println("MENU: Alterando ${usuarioEncontrado.nome} (${usuarioEncontrado.email}")
    println("OPÇÕES:")
    println("[0] Voltar\n[1] Nome [2] Data de Nascimento [3] Sexo [4] Senha")
    when {
        usuarioEncontrado.tipoUsuario == TipoUsuario.ORGANIZADOR -> println(
            "[5] CNPJ [6] Razão Social [7] Nome Fantasia"
        )
    }
    println("Digite opção: ")
    val opcaoAlterarUsuario = readln()
```



```
when (opcaoAlterarUsuario) {  
    "0" -> {  
        println("OK: Selecionado Voltar.\n")  
        menuAlterarUsuario = false  
    }  
  
    "1" -> {  
        print("OK: Digite Nome atualizado: ")  
        usuarioEncontrado.nome = readln().uppercase()  
        println("OK: NOME DEFINIDO '${usuarioEncontrado.nome}.\n")  
    }  
}
```

```

"2" -> {

    // Loop e validação de data de nascimento
    do {
        println("MENU: ALTERAR DATA DE NASCIMENTO")
        print("Digite Somente Dia de Nascimento (DD): ")
        val diaNascimento = readln().toIntOrNull() ?: 0
        print("Digite Somente Mês de Nascimento (MM): ")
        val mesNascimento = readln().toIntOrNull() ?: 0
        print("Digite Somente Ano de Nascimento (AAAA): ")
        val anoNascimento = readln().toIntOrNull() ?: 0

        dataValida = false
        when {
            diaNascimento in 1 .. ≤ 31 && mesNascimento in 1 .. ≤ 12 && anoNascimento in 1920 .. ≤ 2020 -> {
                dataValida = true
                usuarioEncontrado.dataNascimento = "$diaNascimento/$mesNascimento/$anoNascimento"
            }
            else -> println("ERRO: Data inválida. Tente novamente.")
        }
    } while (!dataValida)
    println("OK: DATA DE NASCIMENTO DEFINIDA '${usuarioEncontrado.dataNascimento}'.")
}

```

```

"3" -> {
    println("ALTERANDO: Sexo [1] MASCULINO, [2] FEMININO, [3] NÃO INFORMADO")
    val alterarSexo = readln()
    usuarioEncontrado.sex = 
        when (alterarSexo) {
            "1" -> SexoUsuario.MASCULINO
            "2" -> SexoUsuario.FEMININO
            else -> SexoUsuario.NAO_INFORMADO
        }
    println("OK: SEXO DEFINIDO ${usuarioEncontrado.sex}.\\n")
}

"4" -> {
    // Variável e loop para inserir senha (validar com duas etapas)
    var cadastroSenha: String
    do {
        print("OK: Digite Senha atualizada: ")
        cadastroSenha = readln()
        print("Confirme Senha atualizada: ")
        val confirmarSenha = readln()
        when {
            cadastroSenha.length < 8 -> println("ERRO: A senha deve ter no mínimo 8 caracteres.\\n")
            cadastroSenha != confirmarSenha -> println("ERRO: Senhas não conferem. Tente novamente.")
            else -> println("OK: SENHA ATUALIZADA.\\n")
        }
    }
}

```

```
"5" -> {

    // Validação de usuário (organizador) e loop para validar CNPJ
    when {
        usuarioEncontrado.tipoUsuario == TipoUsuario.ORGANIZADOR -> {
            var cnpjValido = false
            var alterarCNPJ: String
            do {
                print("Digite CNPJ atualizado (14 dígitos): ")
                alterarCNPJ = readln()

                when {
                    alterarCNPJ.length == 14 -> cnpjValido = true
                    else -> println("ERRO: CNPJ inválido. Tente novamente.\n")
                }
            } while (!cnpjValido)

            usuarioEncontrado.cnpj = alterarCNPJ
            println("OK: CNPJ DEFINIDO '${usuarioEncontrado.cnpj}'.\n")
        }
    }

    else -> println("ERRO: Opção inválida. Tente novamente.")
}
}
```

```

    "6" -> {
        when {
            usuarioEncontrado.tipoUsuario == TipoUsuario.ORGANIZADOR -> {
                print("OK: Digite Razão Social atualizada: ")
                usuarioEncontrado.razaoSocial = readln().uppercase()
                println("OK: RAZÃO SOCIAL DEFINIDA '${usuarioEncontrado.razaoSocial}'.\n")
            }

            else -> println("ERRO: Opção inválida. Tente novamente.")
        }
    }

    "7" -> {
        when {
            usuarioEncontrado.tipoUsuario == TipoUsuario.ORGANIZADOR -> {
                print("OK: Digite Nome Fantasia Atualizado: ")
                usuarioEncontrado.nomeFantasia = readln().uppercase()
                println("OK: NOME FANTASIA DEFINIDO '${usuarioEncontrado.nomeFantasia}'.\n")
            }

            else -> println("ERRO: Opção inválida. Tente novamente.")
        }
    }

    else -> println("ERRO: Opção inválida. Tente novamente.")
}

} while (menuAlterarUsuario)
}

```

```

var menuAlterarUsuario = true

// Alterar dados de conta ativa
do {
    println("MENU: Alterando ${usuarioEncontrado.nome} (${usuarioEncontrado.email}")
    println("OPÇÕES:")
    println("[0] Voltar\n[1] Nome [2] Data de Nascimento [3] Sexo [4] Senha")
    when {
        usuarioEncontrado.tipoUsuario == TipoUsuario.ORGANIZADOR -> println(
            "[5] CNPJ [6] Razão Social [7] Nome Fantasia"
        )
    }
    println("Digite opção: ")
    val opcaoAlterarUsuario = readln()
    when (opcaoAlterarUsuario) {
        "0" -> {
            println("OK: Selecionado Voltar.\n")
            menuAlterarUsuario = false
        }
        "1" -> {
            print("OK: Digite Nome atualizado: ")
            usuarioEncontrado.nome = readln().uppercase()
            println("OK: NOME DEFINIDO '${usuarioEncontrado.nome}'.\n")
        }
        "2" -> {
            // Loop e validação de data de nascimento
            do {
                println("MENU: ALTERAR DATA DE NASCIMENTO")
                print("Digite Somente Dia de Nascimento (DD): ")
                val diaNascimento = readln().toIntOrNull() ?: 0
                print("Digite Somente Mês de Nascimento (MM): ")
                val mesNascimento = readln().toIntOrNull() ?: 0
                print("Digite Somente Ano de Nascimento (AAAA): ")
                val anoNascimento = readln().toIntOrNull() ?: 0

                dataValida = false
                when {
                    diaNascimento in 1 .. 31 && mesNascimento in 1 .. 12 && anoNascimento in 1920 .. 2020
                        dataValida = true
                        usuarioEncontrado.dataNascimento = "$diaNascimento/$mesNascimento/$anoNascimento"
                }

                else -> println("ERRO: Data inválida. Tente novamente.")
            }
            while (!dataValida)
            println("OK: DATA DE NASCIMENTO DEFINIDA '${usuarioEncontrado.dataNascimento}'")
        }
        "3" -> {
            println("ALTERANDO: Sexo [1] MASCULINO, [2] FEMININO, [3] NÃO INFORMADO")
            val alterarSexo = readln()
            usuarioEncontrado.sexo =
                when (alterarSexo) {
                    "1" -> SexoUsuario.MASCULINO
                    "2" -> SexoUsuario.FEMININO
                    else -> SexoUsuario.NAO_INFORMADO
                }
            println("OK: SEXO DEFINIDO ${usuarioEncontrado.sexo}.\n")
        }
        "4" -> {
            // Variável e loop para inserir senha (validar com duas etapas)
            var cadastrSenha: String
            do {
                print("OK: Digite Senha atualizada: ")
                cadastrSenha = readln()
                print("Confirme Senha atualizada: ")
                val confirmarSenha = readln()
                when {
                    cadastrSenha.length < 8 -> println("ERRO: A senha deve ter no mínimo 8 caracteres.\n")
                    cadastrSenha != confirmarSenha -> println("ERRO: Senhas não conferem. Tente novamente.")
                    else -> println("OK: SENHA ATUALIZADA.\n")
                }
            }
            while (true)
        }
        "5" -> {
            // Validação de usuário (organizador) e loop para validar CNPJ
            when {
                usuarioEncontrado.tipoUsuario == TipoUsuario.ORGANIZADOR -> {
                    var cnpjValido = false
                    var alterarCNPJ: String
                    do {
                        print("Digite CNPJ atualizado (14 dígitos): ")
                        alterarCNPJ = readln()

                        when {
                            alterarCNPJ.length == 14 -> cnpjValido = true
                            else -> println("ERRO: CNPJ inválido. Tente novamente.\n")
                        }
                    } while (!cnpjValido)

                    usuarioEncontrado.cnpj = alterarCNPJ
                    println("OK: CNPJ DEFINIDO '${usuarioEncontrado.cnpj}'.\n")
                }
                else -> println("ERRO: Opção inválida. Tente novamente.")
            }
        }
        "6" -> {
            when {
                usuarioEncontrado.tipoUsuario == TipoUsuario.ORGANIZADOR -> {
                    println("OK: Digite Razão Social atualizada: ")
                    usuarioEncontrado.razaoSocial = readln().uppercase()
                    println("OK: RAZÃO SOCIAL DEFINIDA '${usuarioEncontrado.razaoSocial}'.\n")
                }
                else -> println("ERRO: Opção inválida. Tente novamente.")
            }
        }
        "7" -> {
            when {
                usuarioEncontrado.tipoUsuario == TipoUsuario.ORGANIZADOR -> {
                    print("OK: Digite Nome Fantasia Atualizado: ")
                    usuarioEncontrado.nomeFantasia = readln().uppercase()
                    println("OK: NOME FANTASIA DEFINIDO '${usuarioEncontrado.nomeFantasia}'.\n")
                }
                else -> println("ERRO: Opção inválida. Tente novamente.")
            }
        }
    }
}
} while (menuAlterarUsuario)

```

Dendê Eventos (Descobrindo Kotlin)

Visão Geral

unex

User Story 7: Alterar Evento

- Objetivo

Um usuário organizador do serviço deve ter acesso a alteração dos próprios eventos. Portanto, deve conseguir realizar alterações em nome, página, descrição, período, tipo (categoria), evento principal vinculado, modalidade, capacidade máxima, local, preço e política/taxa de estorno de ingressos.

Estrutura de dados: Alterar Evento

- Data class

Para armazenar (em memória) dados de eventos e ingressos, são usados data classes (Evento e Ingresso).

```
data class Evento(  
    val id: Int,  
    val organizadorEmail: String,  
    var pagina: String,  
    var nome: String,  
    var descricao: String,  
    var diaInicio: Int, var mesInicio: Int, var anoInicio: Int,  
    var horaInicio: Int, var minutoInicio: Int,  
    var diaTermino: Int, var mesTermino: Int, var anoTermino: Int,  
    var horaTermino: Int, var minutoTermino: Int,  
    var tipo: TipoEvento,  
    var idEventoPrincipal: Int?,  
    var modalidade: ModalidadeEvento,  
    var capacidadeMax: Int,  
    var local: String,  
    var statusEvento: Boolean,  
    var precoIngresso: Double,  
    var aceitaEstorno: Boolean,  
    var taxaEstorno: Double = 0.0  
)
```

Estrutura de dados: Alterar Evento

- Data class

Para armazenar (em memória) dados de eventos e ingressos, são usados data classes (Evento e Ingresso).

```
data class Ingresso(  
    val id: Int,  
    val idEvento: Int,  
    val emailUsuario: String,  
    var statusDisponibilidade: Boolean,  
    val valorPago: Double  
)
```

Estrutura de dados: Alterar Evento

- Enum class

Para dados especificados/predefinidos, são usados enum classes (TipoEvento, ModalidadeEvento).

```
enum class TipoEvento {  
    SOCIAL, CORPORATIVO, ACADEMICO, CULTURAL_ENTRETENIMENTO, RELIGIOSO, ESPORTIVO,  
    FEIRA, CONGRESSO, OFICINA, CURSO, TREINAMENTO, AULA, SEMINARIO, PALESTRA, SHOW,  
    FESTIVAL, EXPOSICAO, RETIRO, CULTO, CELEBRACAO, CAMPEONATO, CORRIDA, OUTRO  
}  
enum class ModalidadeEvento { PRESENCIAL, REMOTO, HIBRIDO }
```

Estrutura de dados: Alterar Evento

- List

Para atualizar os data classes (Evento e Ingresso), são usadas listas mutáveis (listaEventos e listaIngressos).

```
val listaEventos = mutableListOf<Evento>()
val listaIngressos = mutableListOf<Ingresso>()
```

Experiência do Usuário: Alterar Evento

Seus eventos:

ID: 1 | NOME: Aula de Reposição

Digite o ID do evento para alterar: **1**

MENU: EDITANDO Aula de Reposição (1).

OPÇÕES:

[0] Voltar

[1] Nome [2] Página [3] Descrição [4] Período [5] Tipo

[6] Evento Vinculado [7] Modalidade [8] Capacidade [9] Local [10] Preço/Estorno

Digite opção:

7

ALTERANDO: Modalidade [1] PRESENCIAL [2] REMOTO [3] HÍBRIDO

Digite opção: **1**

OK: MODALIDADE DEFINIDA PRESENCIAL.

MENU: EDITANDO Aula de Reposição (1).

OPÇÕES:

[0] Voltar



Regras de negócio, validações: Alterar Evento

- Usuários comuns
não acessam o
menu de alterar
eventos
- Organizadores só
veem os próprios
eventos em lista

```
var possuiEventos = false

// Busca todos os eventos do organizador e lista
when (usuarioEncontrado.tipoUsuario) {
    TipoUsuario.ORGANIZADOR -> {
        println("\nALTERAR EVENTO")
        println("Seus eventos:")
        for (evento in listaEventos) {
            when {
                evento.organizadorEmail == usuarioEncontrado.email -> {
                    println("ID: ${evento.id} | NOME: ${evento.nome}")
                    possuiEventos = true
                }
            }
        }
    }
}
```

Regras de negócio, validações: Alterar Evento

- Organizadores
só editam os
próprios
eventos

```
for (evento in listaEventos) {
    when {
        evento.id == idEvento && evento.organizadorEmail == usuarioEncontrado.email -> eventoAlterando = evento
    }
}

when (eventoAlterando) {
    null -> println("ERRO: Evento inválido. Tente novamente.")
    else -> {

        // Caso o evento esteja desativado, é necessário reativar para alterar dados
        when {
            !eventoAlterando.statusEvento -> {
                println("\nAVISO: Este é um evento desativado. Reativar para alterar? [1] SIM [2] NÃO")
                print("Digite opção: ")
                val reativarEvento = readln()
                when (reativarEvento) {
                    "1" -> {
                        eventoAlterando.statusEvento = true
                        println("OK: Evento reativado. Solicite novamente.\n")
                    }

                    "2" -> println("OK: Operação cancelada.")
                    else -> println("ERRO: Opção inválida. Solicite novamente.")
                }
            }
        }
    }
}
```

Regras de negócio, validações: Alterar Evento

- Período do evento (datas posteriores a atual, duração mínima, etc)

```
when {  
    dataInicio < dataHoje ->  
        println("ERRO: O evento não pode ser no passado.")  
  
    dataFim < dataInicio ->  
        println("ERRO: Data de término antes da data de início.")  
  
    dataFim == dataInicio && minutagemFim < minutagemInicio ->  
        println("ERRO: Hora de término antes da hora de início.")  
  
    dataFim == dataInicio && (minutagemFim - minutagemInicio) < 30 ->  
        println("ERRO: A duração mínima é de 30 minutos.")  
  
    horaInicio !in 0 .. ≤ 23 || horaFinal !in 0 .. ≤ 23 || minutoInicio !in 0 .. ≤ 59 || minutoFinal !in 0 .. ≤ 59 ->  
        println("ERRO: Horário inválido. Use 0-23 para horas e 0-59 para minutos.")  
  
    else -> {
```

Regras de negócio, validações: Alterar Evento

- Vinculação de evento principal a sub-evento

```
when {  
  
    // Se ID do evento principal for igual ao digitado...  
    evento.id == eventoPrincipal &&  
  
    // ... ID do evento principal for diferente do sub-evento...  
    evento.id != eventoAlterando.id &&  
  
    // ... Evento pertence ao organizador...  
    evento.organizadorEmail == usuarioEncontrado.email &&  
  
    // ... Evento estiver ativado...  
    evento.statusEvento ->  
  
    // Então ID do evento principal é válido  
    idValido = true  
}
```

Regras de negócio, validações: Alterar Evento

- Capacidade máxima maior que 0 e número de ingressos vendidos

```
var capacidadeValida = false
var alterarCapacidade: Int
do {
    print("Digite Capacidade Máxima de Pessoas atualizada: ")
    alterarCapacidade = readln().toIntOrNull() ?: 0

    // A capacidade máxima de pessoas não pode ficar menor que a quantidade de ingressos vendidos
    when {
        alterarCapacidade >= ingressosVendidos && alterarCapacidade > 0 -> capacidadeValida = true
        else -> println("ERRO: Capacidade inválida ou menor que ingressos vendidos. Tente novamente.\n")
    }
} while (!capacidadeValida)
eventoAlterando.capacidadeMax = alterarCapacidade
println("OK: CAPACIDADE DEFINIDA ${eventoAlterando.capacidadeMax}. \n")
```

Regras de negócio, validações: Alterar Evento

- Valores de preço e taxa de estorno não podem ser negativos

```
var precoValido = false
var cadastroPreco: Double
do {
    print("Digite Preço do Ingresso atualizado: ")
    cadastroPreco = readln().toDoubleOrNull() ?: 0.0
    when {
        cadastroPreco >= 0 -> precoValido = true
        else -> println("ERRO: Preço inválido. Tente novamente.\n")
    }
} while (!precoValido)  
  
eventoAlterando.precoIngresso = cadastroPreco
println("OK: PREÇO DEFINIDO ${eventoAlterando.precoIngresso}.\n")
```

```
var alterarTaxa: Double
var taxaInvalida: Boolean
do {
    print("Digite Taxa de Estorno (%): ")
    alterarTaxa = readln().toDoubleOrNull() ?: -1.0

    when {
        alterarTaxa !in 0.0 .. ≤ 100.0 -> {
            taxaInvalida = true
            println("ERRO: Taxa inválida. Tente novamente.\n")
        }
        else -> taxaInvalida = false
    }
} while (taxaInvalida)
eventoAlterando.taxaEstorno = alterarTaxa
println("OK: TAXA DE ESTORNO DEFINIDA ${eventoAlterando.taxaEstorno}.\n")
```



Implementação: Alterar Evento

```
"8" -> {
    var possuiEventos = false

    // Busca todos os eventos do organizador e lista
    when (usuarioEncontrado.tipoUsuario) {
        TipoUsuario.ORGANIZADOR -> {
            println("\nALTERAR EVENTO")
            println("Seus eventos:")
            for (evento in listaEventos) {
                when {
                    evento.organizadorEmail == usuarioEncontrado.email -> {
                        println("ID: ${evento.id} | NOME: ${evento.nome}")
                        possuiEventos = true
                    }
                }
            }
        }
    }
}
```

```
// Busca o evento selecionado
when (possuiEventos) {
    false -> println("ERRO: Nenhum evento encontrado.")
    true -> {
        print("Digite o ID do evento para alterar: ")
        val idEvento = readln().toIntOrNull() ?: 0

        // Variável para selecionar o evento a ser alterado
        var eventoAlterando: Evento? = null

        for (evento in listaEventos) {
            when {
                evento.id == idEvento && evento.organizadorEmail == usuarioEncontrado.email ->
                    eventoAlterando = evento
            }
        }
    }
}
```

```

when (eventoAlterando) {
    null -> println("ERRO: Evento inválido. Tente novamente.")
    else -> {

        // Caso o evento esteja desativado, é necessário reativar para alterar dados
        when {
            !eventoAlterando.statusEvento -> {
                println("\nAVISO: Este é um evento desativado. " +
                    "Reativar para alterar? [1] SIM [2] NÃO")
                print("Digite opção: ")
                val reativarEvento = readln()
                when (reativarEvento) {
                    "1" -> {
                        eventoAlterando.statusEvento = true
                        println("OK: Evento reativado. Solicite novamente.\n")
                    }

                    "2" -> println("OK: Operação cancelada.")
                    else -> println("ERRO: Opção inválida. Solicite novamente.")
                }
            }
        }
    }
}

```

```

else -> {
    var menuAlterarEvento = true

    // Menu para alterar os dados do evento
    do {
        println("MENU: EDITANDO ${eventoAlterando.nome} (${eventoAlterando.id}).")
        println("OPÇÕES:")
        println("[0] Voltar\n[1] Nome [2] Página [3] Descrição [4] Período [5] Tipo")
        println("[6] Evento Vinculado [7] Modalidade [8] Capacidade [9] Local [10] Preço/Estorno")
        println("Digite opção: ")
        val opcaoAlterarEvento = readln()

        when (opcaoAlterarEvento) {
            "0" -> {
                println("OK: Selecionado Voltar.\n")
                menuAlterarEvento = false
            }

            "1" -> {
                print("Digite Nome atualizado: ")
                eventoAlterando.nome = readln()
                println("OK: NOME DEFINIDO '${eventoAlterando.nome}'.")
            }
        }
    } while (menuAlterarEvento)
}

```

```
"2" -> {
    print("Digite Página atualizada: ")
    eventoAlterando.pagina = readln()
    println("OK: PÁGINA DEFINIDA '${eventoAlterando.pagina}'.")
}

"3" -> {
    print("Digite Descrição atualizada: ")
    eventoAlterando.descricao = readln()
    println("OK: DESCRIÇÃO DEFINIDA '${eventoAlterando.descricao}'.")
}
```

```
"4" -> {
    var dataValida = false

    do {
        println("\nALTERAR PERÍODO DO EVENTO")

        println("MENU: ALTERAR DATA DE INÍCIO")
        print("Digite Somente Dia atualizado (DD): ")
        val diaInicio = readln().toIntOrNull() ?: 0
        print("Digite Somente Mês atualizado (MM): ")
        val mesInicio = readln().toIntOrNull() ?: 0
        print("Digite Somente Ano atualizado (AAAA): ")
        val anoInicio = readln().toIntOrNull() ?: 0
        print("Digite Somente Hora atualizado (HH): ")
        val horaInicio = readln().toIntOrNull() ?: 0
        print("Digite Somente Minuto atualizado (MM): ")
        val minutoInicio = readln().toIntOrNull() ?: 0

        println("MENU: ALTERAR DATA DE TÉRMINO")
        print("Digite Somente Dia atualizado (DD): ")
        val diaFinal = readln().toIntOrNull() ?: 0
        print("Digite Somente Mês atualizado (MM): ")
        val mesFinal = readln().toIntOrNull() ?: 0
        print("Digite Somente Ano atualizado (AAAA): ")
        val anoFinal = readln().toIntOrNull() ?: 0
        print("Digite Somente Hora atualizado (HH): ")
        val horaFinal = readln().toIntOrNull() ?: 0
        print("Digite Somente Minuto atualizado (MM): ")
        val minutoFinal = readln().toIntOrNull() ?: 0
```

```

val dataInicio = (anoInicio * 10000) + (mesInicio * 100) + diaInicio
val dataFim = (anoFinal * 10000) + (mesFinal * 100) + diaFinal

val minutagemInicio = (horaInicio * 60) + minutoInicio
val minutagemFim = (horaFinal * 60) + minutoFinal

// Validação de data igual a de cadastro de evento
when {
    dataInicio < dataHoje ->
        println("ERRO: O evento não pode ser no passado.")

    dataFim < dataInicio ->
        println("ERRO: Data de término antes da data de início.")

    dataFim == dataInicio && minutagemFim < minutagemInicio ->
        println("ERRO: Hora de término antes da hora de início.")

    dataFim == dataInicio && (minutagemFim - minutagemInicio) < 30 ->
        println("ERRO: A duração mínima é de 30 minutos.")

    horaInicio !in 0 .. ≤ 23 || horaFinal !in 0 .. ≤ 23
        || minutoInicio !in 0 .. ≤ 59 || minutoFinal !in 0 .. ≤ 59 ->
            println("ERRO: Horário inválido. Use 0-23 para horas e 0-59 para minutos.")
}

```

```
        else -> {
            eventoAlterando.diaInicio = diaInicio
            eventoAlterando.mesInicio = mesInicio
            eventoAlterando.anoInicio = anoInicio
            eventoAlterando.horaInicio = horaInicio
            eventoAlterando.minutoInicio = minutoInicio
            eventoAlterando.diaTermino = diaFinal
            eventoAlterando.mesTermino = mesFinal
            eventoAlterando.anoTermino = anoFinal
            eventoAlterando.horaTermino = horaFinal
            eventoAlterando.minutoTermino = minutoFinal
            println("\nOK:\nDATA DE INÍCIO DEFINIDA $diaInicio/$mesInicio/$anoInicio.")
            println("DATA DE TÉRMINO DEFINIDA $diaFinal/$mesFinal/$anoFinal.")
            dataValida = true
        }
    }
} while (!dataValida)
}
```

```

"5" -> {
    println("ALTERANDO: Tipo de evento")
    println("[1] Social [2] Corporativo [3] Acadêmico [4] Cultural/Entretenimento")
    println("[5] Religioso [6] Esportivo [7] Feira [8] Congresso [9] Oficina")
    println("[10] Curso [11] Treinamento [12] Aula [13] Seminário [14] Palestra")
    println("[15] Show [16] Festival [17] Exposição [18] Retiro [19] Culto [20]")
    println("[21] Celebração [22] Campeonato [23] Corrida [24] Outro")
    print("Digite opção: ")
    val alterarTipo = readln()
    eventoAlterando.tipo =
        when (alterarTipo) {
            "1" -> TipoEvento.SOCIAL
            "2" -> TipoEvento.CORPORATIVO
            "3" -> TipoEvento.ACADEMICO
            "4" -> TipoEvento.CULTURAL_ENTRETENIMENTO
            "5" -> TipoEvento.RELIGIOSO
            "6" -> TipoEvento.ESPORTIVO
            "7" -> TipoEvento.FEIRA
            "8" -> TipoEvento.CONGRESSO
            "9" -> TipoEvento.OFICINA
            "10" -> TipoEvento.CURSO
            "11" -> TipoEvento.TREINAMENTO
            "12" -> TipoEvento.AULA
            "13" -> TipoEvento.SEMINARIO
            "14" -> TipoEvento.PALESTRA
        }
    "15" -> TipoEvento.SHOW
    "16" -> TipoEvento.FESTIVAL
    "17" -> TipoEvento.EXPOSICAO
    "18" -> TipoEvento.RETIRO
    "19" -> TipoEvento.CULTO
    "20" -> TipoEvento.CELEBRACAO
    "21" -> TipoEvento.CAMPEONATO
    "22" -> TipoEvento.CORRIDA
    else -> TipoEvento.OUTRO
}
println("OK: TIPO DE EVENTO DEFINIDO ${eventoAlterando.tipo}.\n")

```

```

"6" -> {
    when {
        eventoAlterando.idEventoPrincipal == null -> {
            println("ERRO: Nenhum evento principal vinculado.")
            println("Vincular novo evento a evento principal? [1] SIM [2] NÃO")
        }
    }

    else -> {
        println("Sub-Evento '${eventoAlterando.nome}' | Principal (ID ${eventoAlterando.idEventoPrincipal}).")
        println("Alterar principal? [1] SIM [2] NÃO")
    }
}

print("Digite opção: ")
val vincularPrincipal = readln()

when (vincularPrincipal) {
    "1" -> {
        print("Digite ID do Evento Principal (ou 0 para desvincular): ")
        val eventoPrincipal = readln().toIntOrNull()

        when (eventoPrincipal) {
            0 -> {
                eventoAlterando.idEventoPrincipal = null
                println("OK: EVENTO PRINCIPAL DESVINCULADO.\n")
            }
        }
    }
}

```

```
else -> {
    var isValido = false
    for (evento in listaEventos) {

        // Validações para vincular um evento principal
        when {

            // Se ID do evento principal for igual ao digitado...
            evento.id == eventoPrincipal &&

                // ... ID do evento principal for diferente do sub-evento...
                evento.id != eventoAlterando.id &&

                    // ... Evento pertence ao organizador...
                    evento.organizadorEmail == usuarioEncontrado.email &&

                        // ... Evento estiver ativado...
                        evento.statusEvento ->

                            // Então ID do evento principal é válido
                            isValido = true
        }
    }
}
```

```
when (idValido) {
    true -> {
        eventoAlterando.idEventoPrincipal = eventoPrincipal
        println(
            "OK: ID $eventoPrincipal DEFINIDO COMO " +
            "EVENTO PRINCIPAL DE '${eventoAlterando.nome}'\n"
        )
    }
}

false -> println("ERRO: Vinculação mal-sucedida.")

}
}

"2" -> println("OK: Operação cancelada.")
else -> print("ERRO: Opção inválida. Vinculação mal-sucedida.")
}
```

```

"7" -> {
    println("ALTERANDO: Modalidade [1] PRESENCIAL [2] REMOTO [3] HÍBRIDO")
    print("Digite opção: ")
    val cadastroModalidade = readln()
    eventoAlterando.modalidade =
        when (cadastroModalidade) {
            "1" -> ModalidadeEvento.PRESENCIAL
            "2" -> ModalidadeEvento.REMOTO
            else -> ModalidadeEvento.HIBRIDO
        }
    println("OK: MODALIDADE DEFINIDA ${eventoAlterando.modalidade}.\n")
}

"8" -> {
    var ingressosVendidos = 0
    for (ingresso in listaIngressos) {
        when {
            ingresso.idEvento == eventoAlterando.id && !ingresso.statusDisponibilidade -> ingressosVendidos++
        }
    }
    var capacidadeValida = false
    var alterarCapacidade: Int
    do {
        print("Digite Capacidade Máxima de Pessoas atualizada: ")
        alterarCapacidade = readln().toIntOrNull() ?: 0
    }
}

```

```

    alterarCapacidade = readln().toIntOrNull() ?: 0

    // A capacidade máxima de pessoas não pode ficar menor que a quantidade de ingressos vendidos
    when {
        alterarCapacidade >= ingressosVendidos && alterarCapacidade > 0 -> capacidadeValida = true
        else -> println("ERRO: Capacidade inválida ou menor que ingressos vendidos. Tente novamente.\n")
    }
} while (!capacidadeValida)
eventoAlterando.capacidadeMax = alterarCapacidade
println("OK: CAPACIDADE DEFINIDA ${eventoAlterando.capacidadeMax}. \n")
}

"9" -> {
    print("Digite Local atualizado: ")
    eventoAlterando.local = readln()
    println("OK: LOCAL DEFINIDO ${eventoAlterando.local}.")
}

"10" -> {
    var precoValido = false
    var cadastroPreco: Double
    do {
        print("Digite Preço do Ingresso atualizado: ")
        cadastroPreco = readln().toDoubleOrNull() ?: 0.0
        when {

```

```

    cadastroPreco >= 0 -> precoValido = true
    else -> println("ERRO: Preço inválido. Tente novamente.\n")
}
} while (!precoValido)

eventoAlterando.precoIngresso = cadastroPreco
println("OK: PREÇO DEFINIDO ${eventoAlterando.precoIngresso}.\n")

println("ALTERANDO: Aceita estorno/devolução de ingresso? [1] SIM [2] NÃO")
print("Digite opção: ")
val alterarEstorno = readln()

when (alterarEstorno) {
    "1" -> {
        eventoAlterando.aceitaEstorno = true
        val statusTexto = when (eventoAlterando.aceitaEstorno) {
            true -> "[SIM]"
            false -> "[NÃO]"
        }
        println("OK: ACEITA ESTORNO DEFINIDO $statusTexto.")
    }
}

```

var alterarTaxa: Double
var taxaInvalida: Boolean

```

do {
    print("Digite Taxa de Estorno (%): ")
    alterarTaxa = readln().toDoubleOrNull() ?: -1.0

    when {
        alterarTaxa !in 0.0 .. 100.0 -> {
            taxaInvalida = true
            println("ERRO: Taxa inválida. Tente novamente.\n")
        }

        else -> taxaInvalida = false
    }
} while (taxaInvalida)
eventoAlterando.taxaEstorno = alterarTaxa
println("OK: TAXA DE ESTORNO DEFINIDA ${eventoAlterando.taxaEstorno}.\n")
}

else -> {
    eventoAlterando.aceitaEstorno = false
    val statusTexto = when (eventoAlterando.aceitaEstorno) {
        true -> "[SIM]"
        false -> "[NÃO]"
    }
    eventoAlterando.taxaEstorno = 0.0
    println("OK: ACEITA ESTORNO DEFINIDO $statusTexto.")
}

```



// Opção indisponível para usuários comuns

TipoUsuario.COMUM -> *println("ERRO: Opcão inválida. Tente novamente.")*

User Story 10: Listar eventos do organizador

- Objetivo

Um usuário organizador do serviço deve ter acesso a alteração dos próprios eventos. Portanto, deve conseguir realizar alterações em nome, página, descrição, período, tipo (categoria), evento principal vinculado, modalidade, capacidade máxima, local, preço e política/taxa de estorno de ingressos.

Est. de dados: Listar eventos do organizador

- Data class

Para armazenar (em memória) dados de eventos e ingressos, são usados data classes (Evento e Ingresso).

```
data class Evento(  
    val id: Int,  
    val organizadorEmail: String,  
    var pagina: String,  
    var nome: String,  
    var descricao: String,  
    var diaInicio: Int, var mesInicio: Int, var anoInicio: Int,  
    var horaInicio: Int, var minutoInicio: Int,  
    var diaTermino: Int, var mesTermino: Int, var anoTermino: Int,  
    var horaTermino: Int, var minutoTermino: Int,  
    var tipo: TipoEvento,  
    var idEventoPrincipal: Int?,  
    var modalidade: ModalidadeEvento,  
    var capacidadeMax: Int,  
    var local: String,  
    var statusEvento: Boolean,  
    var precoIngresso: Double,  
    var aceitaEstorno: Boolean,  
    var taxaEstorno: Double = 0.0  
)
```

Est. de dados: Listar eventos do organizador

- Data class

Para armazenar (em memória) dados de eventos e ingressos, são usados data classes (Evento e Ingresso).

```
data class Ingresso(  
    val id: Int,  
    val idEvento: Int,  
    val emailUsuario: String,  
    var statusDisponibilidade: Boolean,  
    val valorPago: Double  
)
```

Est. de dados: Listar eventos do organizador

- List

Para buscar os data classes (Evento e Ingresso), são usadas listas mutáveis (listaEventos e listaIngressos).

```
val listaEventos = mutableListOf<Evento>()
val listaIngressos = mutableListOf<Ingresso>()
```

Exp. do Usuário: Listar eventos organizador

VISUALIZAR EVENTOS

Seus eventos:

ID | STATUS | NOME | PERÍODO | LOCAL | PREÇO | CAPACIDADE

1 | [ATIVO] | Aula de Reposição | 28/2/2026 | Centro Universitário de Excelência - UNEX (Feira de Santana) | R\$ 0.0 | 100

Digite ID para expandir detalhes ou [0] Voltar: [1](#)

Aula de Reposição EXPANDIDO

Nome: Aula de Reposição

Descrição: Aula de reposição referente a 18/02/2026

Página: Aula de reposição referente a 18/02/2026

Início: 28/2/2026 às 15:0

Término: 28/2/2026 às 18:0

Tipo: AULA

Modalidade: PRESENCIAL

Local: Centro Universitário de Excelência - UNEX (Feira de Santana)

Capacidade Máxima: 100

Preço do Ingresso: R\$ 0.0

Aceita Estorno: Sim (Taxa: 0.0%)

Evento Independente.

[QUALQUER TECLA] Voltar.



Regras de negócio, validações: Lista Eventos

- Usuários comuns não acessam opções de usuários organizadores
- Organizadores veem apenas uma lista com seus próprios eventos

```
when (usuarioEncontrado.tipoUsuario) {
    TipoUsuario.ORGANIZADOR -> {
        println("\nVISUALIZAR EVENTOS")

        // Lista (temporária) de eventos do organizador
        val organizadorEventos = mutableListOf<Evento>()
        for (evento in listaEventos) {
            when {
                evento.organizadorEmail == usuarioEncontrado.email) -> organizadorEventos.add(evento)
            }
        }
    }
}
```

Regras de negócio, validações: Lista Eventos

- Organizadores expandem apenas os seus próprios eventos

```
when (opcaoID) {
    0 -> println("OK: Selecionado Voltar.")
    else -> {
        var eventoDetalhes: Evento? = null

        for (evento in listaEventos) {
            when {
                evento.id == opcaoID && evento.organizadorEmail == usuarioEncontrado.email -> eventoDetalhes = event
            }
        }
    }
}
```

Regras de negócio, validações: Lista Eventos

- Listagem de eventos ordenada por período e ordem alfabética

```
// Ordenação da lista por data e nome
organizadorEventos.sortWith(
    comparator = compareBy(
        ...selectors = { it.anoInicio }, { it.mesInicio }, { it.diaInicio }, { it.horaInicio }, { it.nome }
    )
)
```

Implementação: Lista Eventos

```
"7" -> {
    when (usuarioEncontrado.tipoUsuario) {
        TipoUsuario.ORGANIZADOR -> {
            println("\nVISUALIZAR EVENTOS")

            // Lista (temporária) de eventos do organizador
            val organizadorEventos = mutableListOf<Evento>()
            for (evento in listaEventos) {
                when {
                    (evento.organizadorEmail == usuarioEncontrado.email) -> organizadorEventos.add(evento)
                }
            }

            // Ordenação da lista por data e nome
            organizadorEventos.sortWith(
                comparador = compareBy(
                    ...selectors = { it.anoInicio }, { it.mesInicio }, { it.diaInicio }, { it.horaInicio }, { it.nome }
                ))
        }
    }
}

println("Seus eventos:")
println("ID | STATUS | NOME | PERÍODO | LOCAL | PREÇO | CAPACIDADE")

var possuiEventos = false
```

```

for (evento in organizadorEventos) {
    when (evento.organizadorEmail) {
        usuarioEncontrado.email -> {
            val statusTexto = when (evento.statusEvento) {
                true -> "[ATIVO]"
                false -> "[INATIVO]"
            }

            println(
                "${evento.id} | $statusTexto | ${evento.nome} | " +
                "${evento.diaInicio}/${evento.mesInicio}/${evento.anoInicio} | " +
                "${evento.local} | R$ ${evento.precoIngresso} | " +
                "${evento.capacidadeMax}"
            )
        }
        possuiEventos = true
    }
}

when (possuiEventos) {
    false -> println("AVISO: Nenhum evento encontrado.")
}

```

```

// Opção para expandir um evento
true -> {
    print("\nDigite ID para expandir detalhes ou [0] Voltar: ")
    val opcaoID = readln().toIntOrNull() ?: 0

    when (opcaoID) {
        0 -> println("OK: Selecionado Voltar.")
        else -> {
            var eventoDetalhes: Evento? = null

            for (evento in listaEventos) {
                when {
                    evento.id == opcaoID && evento.organizadorEmail == usuarioEncontrado.email -> eventoDetalhes = evento
                }
            }

            when (eventoDetalhes) {
                null -> println("ERRO: Nenhum evento encontrado.")
                else -> {
                    println("\n${eventoDetalhes.nome} EXPANDIDO")
                    println("Nome: ${eventoDetalhes.nome}")
                    println("Descrição: ${eventoDetalhes.descricao}")
                    println("Página: ${eventoDetalhes.pagina}")
                    println(
                        "Inicio: ${eventoDetalhes.diaInicio}/${eventoDetalhes.mesInicio}/${eventoDetalhes.anoInicio} " +
                            "às ${eventoDetalhes.horaInicio}:${eventoDetalhes.minutoInicio}"
                    )
                }
            }
        }
    }
}

```



```
    println(
        "Término: ${eventoDetalhes.diaTermo}/${eventoDetalhes.mesTermo}/${eventoDetalhes.anoTermo} " +
        "às ${eventoDetalhes.horaTermo}:${eventoDetalhes.minutoTermo}"
    )
    println("Tipo: ${eventoDetalhes.tipo}")
    println("Modalidade: ${eventoDetalhes.modalidade}")
    println("Local: ${eventoDetalhes.local}")
    println("Capacidade Máxima: ${eventoDetalhes.capacidadeMax}")
    println("Preço do Ingresso: R$ ${eventoDetalhes.precoIngresso}")

    val textoEstorno = when (eventoDetalhes.aceitaEstorno) {
        true -> "Sim (Taxa: ${eventoDetalhes.taxaEstorno}%)"
        false -> "Não"
    }
    println("Aceita Estorno: $textoEstorno")

    when (eventoDetalhes.idEventoPrincipal) {
        null -> println("Evento Independente.")
        else -> println("Vinculado ao Evento Principal ID: ${eventoDetalhes.idEventoPrincipal}")
    }

    print("[QUALQUER TECLA] Voltar.\n")
    readln()
}
// Opção indisponível para usuários comuns
TipoUsuario.COMUM -> println("ERRO: Opção inválida. Tente novamente.")
```



```
"7" -> {
when (usuarioEncontrado.tipoUsuario) {
    Tipousuario.ORGANIZADOR -> {
        println("\nVISUALIZAR EVENTOS")

        // Lista (temporária) de eventos do organizador
        val organizadorEventos = mutableListOf<Evento>()
        for (evento in listaEventos) {
            when {
                (evento.organizadorEmail == usuarioEncontrado.email) -> organizadorEventos.add(evento)
            }
        }

        // Ordenação da lista por data e nome
        organizadorEventos.sortWith(
            comparador = compareBy(
                ...selectors = { it.anoInicio }, { it.mesInicio }, { it.diaInicio }, { it.horaInicio }, { it.nome }
            )
        )

        println("Seus eventos:")
        println("ID | STATUS | NOME | PERÍODO | LOCAL | PREÇO | CAPACIDADE")

        var possuiEventos = false

for (evento in organizadorEventos) {
    when (evento.organizadorEmail) {
        usuarioEncontrado.email -> {
            val statusTexto = when (evento.statusEvento) {
                true -> "[ATIVO]"
                false -> "[INATIVO]"
            }

            println(
                "${evento.id} | $statusTexto | ${evento.nome} | " +
                    "${evento.diaInicio}/${evento.mesInicio}/${evento.anoInicio} | " +
                    "${evento.local} | R$ ${evento.precoIngresso} | " +
                    "${evento.capacidadeMax}"
            )
        }
    }
}

        possuiEventos = true
    }
}

when (possuiEventos) {
    false -> println("AVISO: Nenhum evento encontrado.")
}
```

```
// Opção para expandir um evento
true -> {
    print("\nDigite ID para expandir detalhes ou [0] Voltar: ")
    val opcaoID = readln().toIntOrNull() ?: 0

    when (opcaoID) {
        0 -> println("OK: Selecionado Voltar.")
        else -> {
            var eventoDetalhes: Evento? = null

            for (evento in listaEventos) {
                when {
                    evento.id == opcaoID && evento.organizadorEmail == usuarioEncontrado.email -> eventoDetalhes = evento
                }
            }

            when (eventoDetalhes) {
                null -> println("ERRO: Nenhum evento encontrado.")
                else -> {
                    println("\n${eventoDetalhes.nome} EXPANDIDO")
                    println("Nome: ${eventoDetalhes.nome}")
                    println("Descrição: ${eventoDetalhes.descricao}")
                    println("Página: ${eventoDetalhes.pagina}")
                    println(
                        "Início: ${eventoDetalhes.diaInicio}/${eventoDetalhes.mesInicio}/${eventoDetalhes.anoInicio} " +
                            "às ${eventoDetalhes.horaInicio}:${eventoDetalhes.minutoInicio}"
                    )
                    println(
                        "Término: ${eventoDetalhes.diaTermino}/${eventoDetalhes.mesTermino}/${eventoDetalhes.anoTermino} " +
                            "às ${eventoDetalhes.horaTermino}:${eventoDetalhes.minutoTermino}"
                    )
                    println("Tipo: ${eventoDetalhes.tipo}")
                    println("Modalidade: ${eventoDetalhes.modalidade}")
                    println("Local: ${eventoDetalhes.local}")
                    println("Capacidade Máxima: ${eventoDetalhes.capacidadeMax}")
                    println("Preço do Ingresso: R$ ${eventoDetalhes.precoIngresso}")

                    val textoEstorno = when (eventoDetalhes.aceiteEstorno) {
                        true -> "Sim (Taxa: ${eventoDetalhes.taxaEstorno}%)"
                        false -> "Não"
                    }
                    println("Aceita Estorno: $textoEstorno")

                    when (eventoDetalhes.idEventoPrincipal) {
                        null -> println("Evento Independente")
                        else -> println("Vinculado ao Evento Principal ID: ${eventoDetalhes.idEventoPrincipal}")
                    }
                }
            }
            print("QUALQUER TECLA Voltar.\n")
            readin()
        }
    }
}

// Opção indisponível para usuários comuns
TiposUsuario.COMUM -> println("ERRO: Opção inválida. Tente novamente.")
```

Visão Geral

User Story 14: Listar ingressos

- Objetivo

Um usuário comum do serviço deve ter acesso a listagem dos próprios ingressos comprados. Ademais, deve conseguir visualizar todos os detalhes do evento de um ingresso da lista.

Estrutura de dados: Listar ingressos

- Data class

Para armazenar (em memória) dados de eventos e ingressos, são usados data classes (Evento e Ingresso).

```
data class Evento(  
    val id: Int,  
    val organizadorEmail: String,  
    var pagina: String,  
    var nome: String,  
    var descricao: String,  
    var diaInicio: Int, var mesInicio: Int, var anoInicio: Int,  
    var horaInicio: Int, var minutoInicio: Int,  
    var diaTermino: Int, var mesTermino: Int, var anoTermino: Int,  
    var horaTermino: Int, var minutoTermino: Int,  
    var tipo: TipoEvento,  
    var idEventoPrincipal: Int?,  
    var modalidade: ModalidadeEvento,  
    var capacidadeMax: Int,  
    var local: String,  
    var statusEvento: Boolean,  
    var precoIngresso: Double,  
    var aceitaEstorno: Boolean,  
    var taxaEstorno: Double = 0.0  
)
```

Estrutura de dados: Listar ingressos

- List

Para buscar os data classes (Evento e Ingresso), são usadas listas mutáveis (listaEventos e listaIngressos).

```
val listaEventos = mutableListOf<Evento>()
```

```
val listaIngressos = mutableListOf<Ingresso>()
```

Experiência do Usuário: Listar ingressos

VISUALIZAR INGRESSOS

ID: 1 | Evento: Aula de Reposição | Data: 28/2/2026 às 15:0 | Valor: R\$ 0.0 | Status: [OK]

ID: 2 | Evento: Aula de Reposição | Data: 28/2/2026 às 15:0 | Valor: R\$ 0.0 | Status: [CANCELADO]

Digite ID do ingresso para expandir/cancelar (ou [0] Voltar): **1**

DETALHES DO INGRESSO

ID: 1

Evento: Aula de Reposição

Data: 28/2/2026 às 15:0

Local: Centro Universitário de Excelência - UNEX (Feira de Santana)

Valor Pago: R\$ 0.0

Status Atual: OK



Regras de negócio, validações: Ver ingressos

- Usuários organizadores não acessam opções de usuários comuns

```
when (usuarioEncontrado.tipoUsuario) {  
    TipoUsuario.COMUM -> {  
        println("\nVISUALIZAR INGRESSOS")  
    }  
}
```

Regras de negócio, validações: Ver ingressos

- Listagem de ingressos ordenada por período e ordem alfabética

```
eventosOrdenados.sortWith(  
    comparator = compareBy(  
        ...selectors = { it.anoInicio }, { it.mesInicio }, { it.diaInicio }, { it.horaInicio }, { it.nome }  
    ))
```

Regras de negócio, validações: Ver ingressos

- Listagem de ingressos:
ativos e futuros
acima
- Somente exibir ingressos do usuário

```
for (evento in eventosOrdenados) {  
    val dataEvento = (evento.anoInicio * 10000) + (evento.mesInicio * 100) + evento.diaInicio  
    when {  
        evento.statusEvento && dataEvento >= dataHoje -> {  
            for (ingresso in listaIngressos) {  
                when {  
                    ingresso.emailUsuario == usuarioEncontrado.email &&  
                    ingresso.idEvento == evento.id &&  
                    !ingresso.statusDisponibilidade -> {  
                        println(  
                            "ID: ${ingresso.id} | Evento: ${evento.nome} | "  
                            "Data: ${evento.diaInicio}/${evento.mesInicio}/${evento.anoInicio} " +  
                            "às ${evento.horaInicio}:${evento.minutoInicio} " +  
                            "| Valor: R$ ${ingresso.valorPago} | Status: [OK]"  
                        )  
                        possuiIngressos = true  
                    }  
                }  
            }  
        }  
    }  
}
```

Regras de negócio, validações: Ver ingressos

- Listagem de ingressos: cancelados e inativos abaixo
- Somente exibir ingressos do usuário

```
for (evento in eventosOrdenados) {  
    val dataEvento = (evento.anoInicio * 10000) + (evento.mesInicio * 100) + evento.diaInicio  
    for (ingresso in listaIngressos) {  
        when {  
            ingresso.emailUsuario == usuarioEncontrado.email && ingresso.idEvento == evento.id -> {  
                when {  
                    ingresso.statusDisponibilidade || !evento.statusEvento || dataEvento < dataHoje -> {  
                        val statusTexto = when (ingresso.statusDisponibilidade) {  
                            true -> "[CANCELADO]"  
                            false -> "[REALIZADO/INATIVO]"  
                        }  
                        println(  
                            "ID: ${ingresso.id} | Evento: ${evento.nome} | " +  
                            "Data: ${evento.diaInicio}/${evento.mesInicio}/${evento.anoInicio} " +  
                            "às ${evento.horaInicio}:${evento.minutoInicio} " +  
                            "| Valor: R$ ${ingresso.valorPago} | Status: $statusTexto"  
                        )  
                        possuiIngressos = true  
                    }  
                }  
            }  
        }  
    }  
}
```

Dendê Eventos (Descobrindo Kotlin)

Regras de negócio, validações: Ver ingressos

- Somente expandir ingressos do usuário

```
var ingressoExpandido: Ingresso? = null
for (ingresso in listaIngressos) {
    when {
        ingresso.id == idIngresso && ingresso.emailUsuario == usuarioEncontrado.email -> ingressoExpandido = ingresso
    }
}
```



Implementação: Listar ingressos

```
"5" -> {
    when (usuarioEncontrado.tipoUsuario) {
        TipoUsuario.COMUM -> {
            println("\nVISUALIZAR INGRESSOS")
            var possuiIngressos = false

            // Lista (temporária) de eventos com ingresso(s)
            val eventosOrdenados = mutableListOf<Evento>()
            for (evento in listaEventos) {
                eventosOrdenados.add(evento)
            }

            // Ordenação da lista por data e nome
            eventosOrdenados.sortWith(
                comparator = compareBy(
                    ...selectors = { it.anoInicio }, { it.mesInicio }, { it.diaInicio }, { it.horaInicio }, { it.nome }
                ))
        }

        // Exibe ingressos ativos e futuros
        for (evento in eventosOrdenados) {
            val dataEvento = (evento.anoInicio * 10000) + (evento.mesInicio * 100) + evento.diaInicio
            when {
                evento.statusEvento && dataEvento >= dataHoje -> {

```

```

        for (ingresso in listaIngressos) {
            when {
                ingresso.emailUsuario == usuarioEncontrado.email &&
                    ingresso.idEvento == evento.id &&
                    !ingresso.statusDisponibilidade -> {
                    println(
                        "ID: ${ingresso.id} | Evento: ${evento.nome} | " +
                        "Data: ${evento.diaInicio}/${evento.mesInicio}/${evento.anoInicio} " +
                        "às ${evento.horaInicio}:${evento.minutoInicio} " +
                        "| Valor: R$ ${ingresso.valorPago} | Status: [OK]"
                    )
                    possuiIngressos = true
                }
            }
        }
    }

    // Exibe ingressos cancelados e inativos
    for (evento in eventosOrdenados) {
        val dataEvento = (evento.anoInicio * 10000) + (evento.mesInicio * 100) + evento.diaInicio
        for (ingresso in listaIngressos) {
            when {
                ingresso.emailUsuario == usuarioEncontrado.email && ingresso.idEvento == evento.id -> {

```

```

    when {
        ingresso.statusDisponibilidade || !evento.statusEvento || dataEvento < dataHoje -> {
            val statusTexto = when (ingresso.statusDisponibilidade) {
                true -> "[CANCELADO]"
                false -> "[REALIZADO/INATIVO]"
            }
            println(
                "ID: ${ingresso.id} | Evento: ${evento.nome} | " +
                    "Data: ${evento.diaInicio}/${evento.mesInicio}/${evento.anoInicio} " +
                    "às ${evento.horaInicio}:${evento.minutoInicio} " +
                    "| Valor: R$ ${ingresso.valorPago} | Status: $statusTexto"
            )
            possuiIngressos = true
        }
    }
}

when (possuiIngressos) {
    false -> println("AVISO: Você não possui ingressos.")

    // Possibilita expandir um ingresso
    true -> {

```

```
print("\nDigite ID do ingresso para expandir/cancelar (ou [0] Voltar): ")
val idIngresso = readln().toIntOrNull() ?: 0

when (idIngresso) {
    0 -> println("OK: Voltando ao Menu Principal.")
    else -> {
        var ingressoExpandido: Ingresso? = null
        for (ingresso in listaIngressos) {
            when {
                ingresso.id == idIngresso && ingresso.emailUsuario == usuarioEncontrado.email ->
                    ingressoExpandido = ingresso
            }
        }
    }
}

when (ingressoExpandido) {
    null -> println("ERRO: Ingresso inexistente ou indisponível.")
    else -> {
        var eventoDoIngresso: Evento? = null
        for (evento in listaEventos) {
            when (evento.id) {
                ingressoExpandido.idEvento -> eventoDoIngresso = evento
            }
        }
    }
}

when (eventoDoIngresso) {
```

```

        null -> println("ERRO: Ingresso inexistente ou indisponível.")
    else -> {
        var eventoDoIngresso: Evento? = null
        for (evento in listaEventos) {
            when (evento.id) {
                ingressoExpandido.idEvento -> eventoDoIngresso = evento
            }
        }
    }

    when (eventoDoIngresso) {
        null -> println("ERRO: Nenhum evento encontrado.")
        else -> {
            println("\nDETALHES DO INGRESSO")
            println("ID: ${ingressoExpandido.id}")
            println("Evento: ${eventoDoIngresso.nome}")
            println(
                "Data: ${eventoDoIngresso.diaInicio}/${eventoDoIngresso.mesInicio}/${eventoDoIngresso.anoInicio} " +
                "às ${eventoDoIngresso.horaInicio}:${eventoDoIngresso.minutoInicio}"
            )
            println("Local: ${eventoDoIngresso.local}")
            println("Valor Pago: R$ ${ingressoExpandido.valorPago}")

            val statusAtual = when (ingressoExpandido.statusDisponibilidade) {
                true -> "CANCELADO"
                false -> "OK"
            }
            println("Status Atual: $statusAtual")
        }
    }
}

```

5" -> {

```
when (usuarioEncontrado.tipoUsuario) {
    TipoUsuario.COMUM -> {
        println("\nVISUALIZAR INGRESSOS")
        var possuiIngressos = false

        // Lista (temporária) de eventos com ingresso(s)
        val eventosOrdenados = mutableListOf<Evento>()
        for (evento in listaEventos) {
            eventosOrdenados.add(evento)
        }

        // Ordenação da lista por data e nome
        eventosOrdenados.sortWith(
            comparador = compareBy(
                ...seletores = { it.anoInicio }, { it.mesInicio }, { it.diaInicio }, { it.horaInicio }, { it.nome } )
        )

        // Exibe ingressos ativos e futuros
        for (evento in eventosOrdenados) {
            val dataEvento = (evento.anoInicio * 10000) + (evento.mesInicio * 100) + evento.diaInicio
            when {
                evento.statusEvento && dataEvento > dataHoje -> {

                    for (ingresso in listaIngressos) {
                        when {
                            ingresso.emailUsuario == usuarioEncontrado.email &&
                                ingresso.idEvento == evento.id &&
                                !ingresso.statusDisponibilidade -> {
                                    println(
                                        "ID: ${ingresso.id} | Evento: ${evento.nome} | +
                                            \"Data: ${evento.diaInicio}/${evento.mesInicio}/${evento.anoInicio}\" |
                                            \"às ${evento.horaInicio}:${evento.minutoInicio}\\" +
                                            "| Valor: R$ ${ingresso.valorPago} | Status: [OK]"
                                    )
                                    possuiIngressos = true
                                }
                        }
                    }
                }
            }
        }

        // Exibe ingressos cancelados e inativos
        for (evento in eventosOrdenados) {
            val dataEvento = (evento.anoInicio * 10000) + (evento.mesInicio * 100) + evento.diaInicio
            for (ingresso in listaIngressos) {
                when {
                    ingresso.emailUsuario == usuarioEncontrado.email && ingresso.idEvento == evento.id &&
                    !ingresso.statusDisponibilidade -> {
                        println(
                            "ID: ${ingresso.id} | Evento: ${evento.nome} | +
                                \"Data: ${evento.diaInicio}/${evento.mesInicio}/${evento.anoInicio}\" |
                                \"às ${evento.horaInicio}:${evento.minutoInicio}\\" +
                                "| Valor: R$ ${ingresso.valorPago} | Status: [OK]"
                        )
                        possuiIngressos = true
                    }
                }
            }
        }
    }
}

// Exibe ingressos cancelados e inativos
for (evento in eventosOrdenados) {
    val dataEvento = (evento.anoInicio * 10000) + (evento.mesInicio * 100) + evento.diaInicio
    for (ingresso in listaIngressos) {
        when {
            ingresso.emailUsuario == usuarioEncontrado.email && ingresso.idEvento == evento.id &&
            !ingresso.statusDisponibilidade -> {
                println(
                    "ID: ${ingresso.id} | Evento: ${evento.nome} | +
                        \"Data: ${evento.diaInicio}/${evento.mesInicio}/${evento.anoInicio}\" |
                        \"às ${evento.horaInicio}:${evento.minutoInicio}\\" +
                        "| Valor: R$ ${ingresso.valorPago} | Status: [OK]"
                )
                possuiIngressos = true
            }
        }
    }
}
```

when (ingresso.statusDisponibilidade || evento.statusEvento || dataEvento < dataHoje -> {
 val statusTexto = when (ingresso.statusDisponibilidade) {
 true -> "[CANCELADO]"
 false -> "[REALIZADO/INATIVO]"
 }
 println(
 "ID: \${ingresso.id} | Evento: \${evento.nome} | +
 \"Data: \${evento.diaInicio}/\${evento.mesInicio}/\${evento.anoInicio}\" |
 \"às \${evento.horaInicio}:\${evento.minutoInicio}\\" +
 "| Valor: R\$ \${ingresso.valorPago} | Status: \$statusTexto"
)
 possuiIngressos = true
}

when (possuiIngressos) {
 false -> println("AVISO: Você não possui ingressos.")

 // Possibilita expandir um ingresso
 true -> {

 print("\nDigite ID do ingresso para expandir/cancelar (ou [0] Voltar): ")
 val idIngresso = readln().toIntOrNull() ?: 0

 when (idIngresso) {
 0 -> println("OK: Voltando ao Menu Principal.")
 else -> {
 var ingressoExpandido: Ingresso? = null
 for (ingresso in listaIngressos) {
 when {
 ingresso.id == idIngresso && ingresso.emailUsuario == usuarioEncontrado.email -> {
 ingressoExpandido = ingresso
 }
 }
 }

 when (ingressoExpandido) {
 null -> println("ERRO: Ingresso inexistente ou indisponível.")
 else -> {
 var eventoDoIngresso: Evento? = null
 for (evento in listaEventos) {
 when (evento.id) {
 ingressoExpandido.idEvento -> eventoDoIngresso = evento
 }
 }

 when (eventoDoIngresso) {
 null -> println("ERRO: Ingresso inexistente ou indisponível.")
 else -> {
 var eventoDoIngresso: Evento? = null
 for (evento in listaEventos) {
 when (evento.id) {
 ingressoExpandido.idEvento -> eventoDoIngresso = evento
 }
 }

 when (eventoDoIngresso) {
 null -> println("ERRO: Ingresso inexistente ou indisponível.")
 else -> {
 println("Detalhes do ingresso:")
 println("ID: \${ingresso.id}")
 println("Evento: \${eventoDoIngresso.nome}")
 println("Data: \${eventoDoIngresso.diaInicio}/\${eventoDoIngresso.mesInicio}/\${eventoDoIngresso.anoInicio}")
 println("Local: \${eventoDoIngresso.local}")
 println("Valor Pago: R\$ \${ingressoExpandido.valorPago}")

 val statusAtual = when (ingressoExpandido.statusDisponibilidade) {
 true -> "CANCELADO"
 false -> "OK"
 }
 println("Status Atual: \$statusAtual")
 }
 }
 }
 }
 }
 }
 }
 }
 }
}

Visão Geral

unex

Visão Geral

unex

Referências

- [OAT 1 - Descobrindo Kotlin - Google Docs](#)
- [ATIVIDADE DE SONDAGEM 1 - Google Docs](#)
- [ATIVIDADE DE SONDAGEM 2 - Google Docs](#)
- [ATIVIDADE DE SONDAGEM 3 - Google Docs](#)
- [Lista de Exercícios 1: Variáveis, Entrada e Saída - Google Docs](#)
- [Lista de Exercícios 3: Introdução a Condicionais - Google Docs](#)
- [Lista de Exercícios 3: Loops - Google Docs](#)