GESFUT

Documento de Arquitetura de Software

Versão 1.6

**Autores**

Bruno Arthur Führ

Romeu Klering

Histórico de Revisões

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versão** | **Descrição** | **Autor** |
| **02/05/2019** | 1.0 | Versão inicial | Bruno Arthur Führ |
| 25/05/2019 | 1.1 | Correções nos tópicos 4, 6 e 7 | Bruno Arthur Führ |
| 04/05/2019 | 1.2 | Preenchimento dos dados do tópico 5 | Bruno Arthur Führ |
| 08/05/2019 | 1.3 | Preenchimento dos dados do tópico 5 | Bruno Arthur Führ |
| 09/05/2019 | 1.4 | Preenchimento dos dados do tópico 5 | Bruno Arthur Führ |
| 10/06/2019 | 1.5 | Atualização dos dados do tópico 5 e tópico 8 | João Francisco Siebel |
| 10/09/2019 | 1.6 | Atualização dos tópicos | Romeu Klering |

Sumário

[**Introdução**](#_heading=h.gjdgxs) **5**

[Finalidade](#_heading=h.30j0zll) 5

[Escopo](#_heading=h.1fob9te) 5

[Definições, Acrônimos e Abreviações](#_heading=h.3znysh7) 5

[Referência](#_heading=h.2et92p0) 5

[**Representação da Arquitetura**](#_heading=h.tyjcwt) **5**

[**Metas e Restrições de Arquitetura**](#_heading=h.3dy6vkm) **6**

[**Visão de Casos de Uso**](#_heading=h.1t3h5sf) **6**

[**Visão Lógica**](#_heading=h.4d34og8) **7**

[Divisão em Pacotes](#_heading=h.2s8eyo1) 7

[Pacote Controller](#_heading=h.17dp8vu) 8

[Pacote Boundary](#_heading=h.3rdcrjn) 11

[Pacote Ouvidoria](#_heading=h.26in1rg) 11

[Pacote Atleta](#_heading=h.lnxbz9) 13

[Pacote Jurídico](#_heading=h.35nkun2) 16

[Pacote Ingressos](#_heading=h.1ksv4uv) 17

[Pacote Entity](#_heading=h.44sinio) 18

[Pacote Ouvidoria](#_heading=h.2jxsxqh) 19

[Pacote Atleta](#_heading=h.z337ya) 20

[Pacote Jurídico](#_heading=h.3j2qqm3) 24

[Pacote Ingressos](#_heading=h.1y810tw) 25

[**Visão de Implantação**](#_heading=h.4i7ojhp) **26**

[**Visão de Implementação**](#_heading=h.2xcytpi) **26**

[Camadas](#_heading=h.1ci93xb) 27

[Soluções utilizadas](#_heading=h.3whwml4) 27

[Convenções](#_heading=h.2bn6wsx) 28

[**Visão de Dados**](#_heading=h.qsh70q) **28**

[Modelo de objetos persistentes](#_heading=h.3as4poj) 29

[Estratégias](#_heading=h.1pxezwc) 29

[Modelo relacional](#_heading=h.49x2ik5) 30

# Introdução

## Finalidade

Este documento oferece uma visão geral arquitetural abrangente do sistema, usando diversas visões arquiteturais para representar diferentes aspectos do sistema. O objetivo deste documento é capturar e comunicar as decisões arquiteturais significativas que foram tomadas em relação ao sistema.

## Escopo

Este documento de Arquitetura de Software se aplica ao sistema **GESFUT** que trata de um sistema de gestão para clubes de futebol.

## Definições, Acrônimos e Abreviações

Aqui estão presentes as definições de todos os termos, acrônimos e abreviações necessários para a correta interpretação do Documento de Arquitetura de Software.

* **MongoDB** – Banco de Dados Orientado a Documentos
* **HTTP** – Hypertext Transfer Protocol
* **TCP -** Transmission Control Protocol
* **IP -** Internet Protocol
* **JSP** – Java Server Pages
* **CSS** – Cascading Style Sheets
* **JSON** – JavaScript Object Notation
* **DAO** – Data Access Object
* **SGBD** – Sistema Gerenciador de Banco de Dados
* **XML** – Extensive Markup Language
* **HTML -** Hypertext Markup Language
* **Base64** – formato de encriptação de dados

## Referência

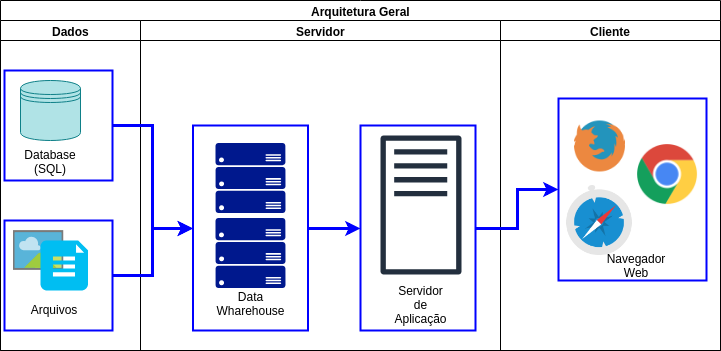
Documentos de referência para este artefato da documentação.

|  |  |  |
| --- | --- | --- |
| **Título** | **Link** | **Data** |
| Resumo de Casos de Uso | casos\_uso.doc | 07/05/2019 |

# Representação da Arquitetura

Esta seção descreve qual é a arquitetura de software do sistema atual e como ela é representada. Nas Visões de Casos de Uso, Lógica, do Processo, de Implantação e de Implementação, este documento enumera as visões necessárias e, para cada uma delas, explica os tipos de elementos do modelo que contém.

Este documento apresenta a arquitetura como uma série de visões: visão de casos de uso, visão lógica, visão de processos, visão de implantação e visão de implementação. Essas visões são apresentadas como Modelos do Astah e utilizam a Linguagem Unificada de Modelagem (UML).



# Metas e Restrições de Arquitetura

Esta seção descreve os requisitos de software e os objetivos que têm um impacto significativo na arquitetura, como proteção, segurança, privacidade, uso de um produto desenvolvido internamente e adquirido pronto para ser usado, portabilidade, distribuição e reutilização.

Existem algumas restrições de requisito e de sistema principais que têm uma relação significativa com a arquitetura. São elas:

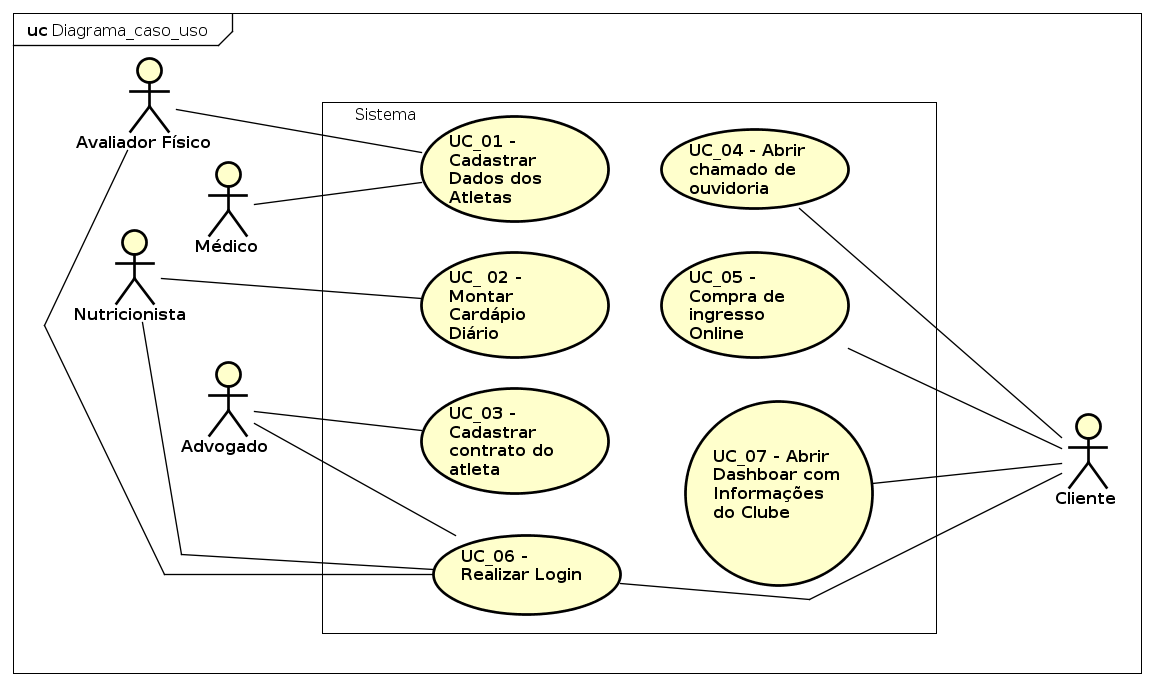
* O sistema deverá ser acessado por meio da Internet;
* As linguagens de desenvolvimento utilizadas para o sistema serão o JavaScript e o TypeScript;
* O Servidor de Aplicação definido para o sistema será feito em Node;
* O Sistema Operacional que dará suporte aos serviços de data warehouse deverá ser LINUX;
* O Sistema Operacional que dará suporte aos serviços da aplicação deverá ser o LINUX;
* O Sistema Gerenciador de Banco de Dados escolhido para suportar a aplicação será o MongoDB;
* Cliente e servidor irão conversar via REST (Web Services);
* O servidor de aplicação deve ser escalável em períodos de compra de ingressos;
* O Client da aplicação será desenvolvido utilizando a tecnologia Angular.

# Visão de Casos de Uso

Esta seção lista os casos de uso ou cenários do modelo de casos de uso que representam uma funcionalidade central e significativa do sistema final ou se têm uma ampla cobertura de arquitetura, ou seja, que experimenta vários elementos de arquitetura e enfatizam ou ilustram um determinado ponto complexo da arquitetura.

Os Casos de Uso do sistema GESFUT são listados a seguir. Os Casos de Uso em negrito são destacados por que são muito importante para a arquitetura:

* **Caso de uso 01 - Cadastrar dados dos atletas**
* Caso de uso 02 - Montar cardápio diário
* Caso de uso 03 - Cadastrar contrato do atleta
* Caso de uso 04 - Abrir chamado de ouvidoria
* **Caso de uso 05 - Compra de ingresso online**
* Caso de uso 06 - Realizar Login
* Caso de uso 07 - Abrir Dashboard com Informações do Clube

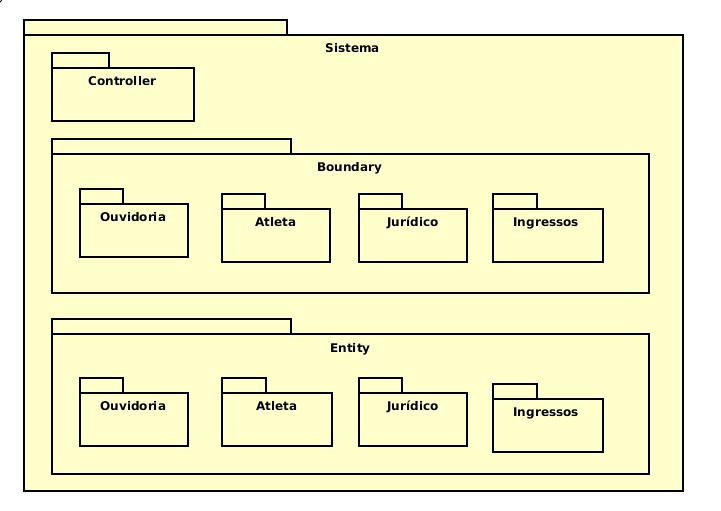


# Visão Lógica

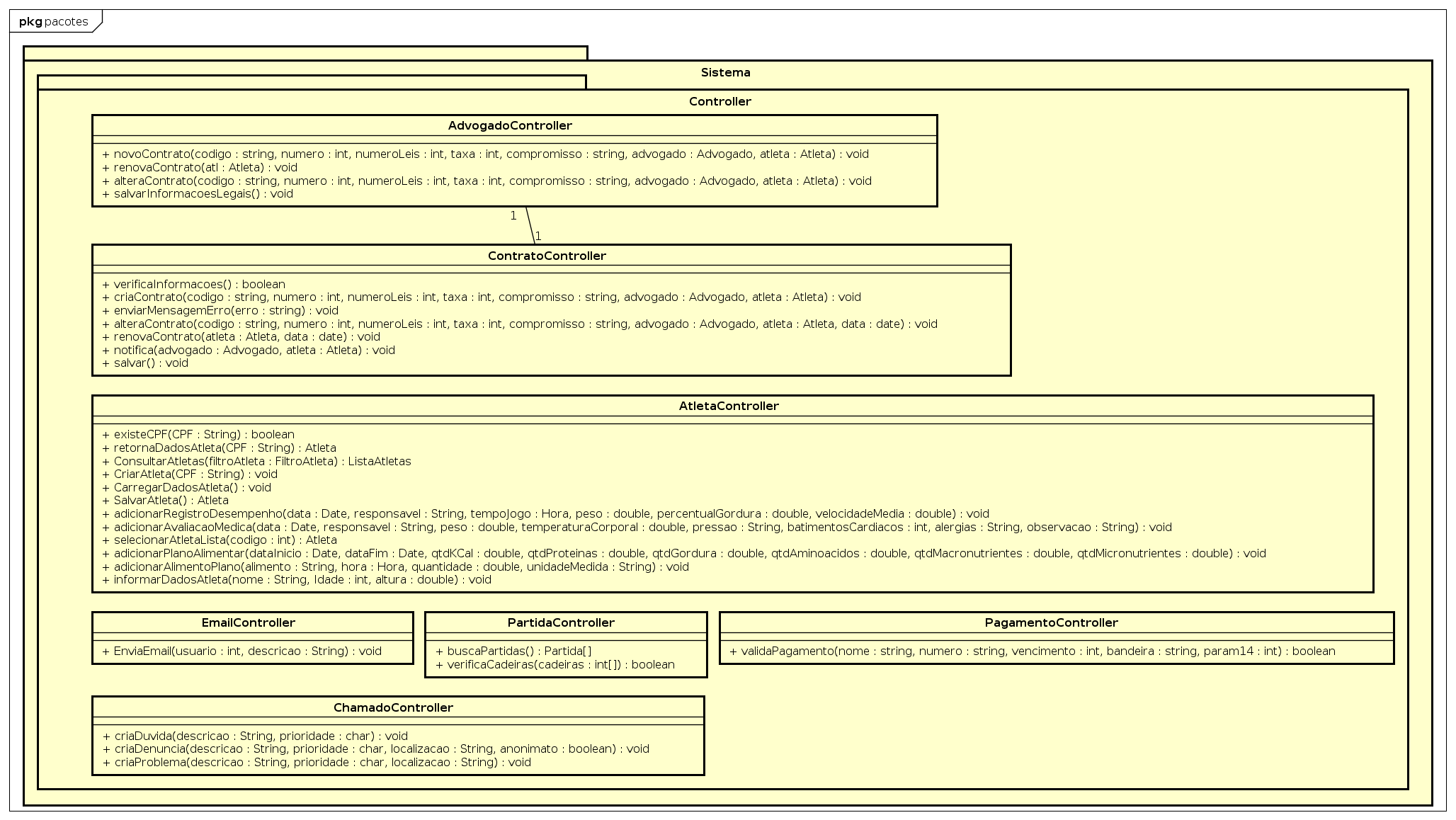
Esta seção descreve as partes significativas do ponto de vista da arquitetura do modelo de design, como sua divisão em subsistemas e pacotes. Além disso, para cada pacote significativo, ela mostra sua divisão em classes e utilitários de classe. Apresenta as classes significativas do ponto de vista da arquitetura e descreve suas responsabilidades, bem como alguns relacionamentos, operações e atributos de grande importância.

## Divisão em Pacotes

Nesta seção serão apresentadas as camadas da arquitetura proposta. Serão descritas as responsabilidades de cada camada quais tecnologias devem ser aplicadas a cada uma delas.



### Pacote Controller



Descrição das classes.

|  |  |
| --- | --- |
| **Classe AtletaController** | |
| Descrição | Esta classe representa o controller da entity Atleta e seus relacionados |
| Relações | Possui relação com as boundaries FormConsultaAtletas, FormAvaliacaoMedica, FormCadastroAtleta e FormAvaliacaoDesempenho Também tem relação com a classe Atleta |
| Responsabilidades | A principal responsabilidade desta classe é realizar o controle dos acessos e manutenções aos dados da classe Atleta e suas filhas |
| Métodos | + existeCPF(CPF : String) : boolean: verifica no banco de dados se existe um atleta com o CPF informado e retorna verdadeiro caso exista  + retornaDadosAtleta(CPF : String) : Atleta: busca um atleta pelo cpf informado no banco de dados, instancia e o retorna  + SalvarAtleta() : void: salva as informações do atleta em edição no controller  + ConsultarAtletas(filtroAtleta : FiltroAtleta) : ListaAtletas recebe um filtro de atletas, consulta os atletas que comportam ao filtro e retorna a lista dos atletas encontrados  + CriarAtleta(CPF : String) : Atleta: cria o obojeto do atleta com o CPF informado e retorna  + adicionarRegistroDesempenho(data : Date, responsavel : String, tempoJogo : Hora, peso : double, percentualGordura : double, velocidadeMedia : double) : void  + adicionarAvaliacaoMedica(data : Date, responsavel : String, peso : double, temperaturaCorporal : double, pressao : String, batimentosCardiacos : int, alergias : String, observacao : String) : void  + selecionarAtletaLista(codigo : int) : Atleta: seleciona, no controller, um atleta presente na lista que possua o código informado  + adicionarPlanoAlimentar(dataInicio : Date, dataFim : Date, qtdKCal : double, qtdProteinas : double, qtdGordura : double, qtdAminoacidos : double, qtdMacronutrientes : double, qtdMicronutrientes : double) : void  + adicionarAlimentoPlano(alimento : String, hora : Hora, quantidade : double, unidadeMedida : String) : void  + informarDadosAtleta(nome : String, Idade : int, altura : double) : void |
| Atributos | atleta: objeto do atleta |

|  |  |
| --- | --- |
| **Classe AdvogadoController** | |
| Descrição | Esta classe representa o controller que integra o form de informações legais e também do form de cadastro |
| Relações | Possui relação com as boundaries informacoesLegaisForm e contratoForm |
| Responsabilidades | A principal responsabilidade desta classe é realizar a integração entre as informações recebidas pelo form de informações legais e o form de contrato. |
| Métodos | + novoContrato(codigo : string, numero : int, numeroLeis : int, taxa : int, compromisso : string, advogado : Advogado, atleta : Atleta) : void: Utiliza as informações passadas pelos form para o controller e as transfere para o controller do contrato  + renovaContrato(atleta: Atleta) : Void: Renova o contrato de um atleta  + alteraContrato(codigo : string, numero : int, numeroLeis : int, taxa : int, compromisso : string, advogado : Advogado, atleta : Atleta) : void: Altera as informações de um contrato de um atleta |
| Atributos |  |

|  |  |
| --- | --- |
| **Classe ContratoController** | |
| Descrição | Esta classe representa o controller do contrato que, por sua vez, realizará ações relacionadas ao contrato |
| Relações | Possui relação com as boundaries advogadoController e a entity contrato |
| Responsabilidades | A principal responsabilidade desta classe é realizar a criação, alteração e renovação de contratos. O controller também possui a responsabilidade de verificar se as informações inseridas estão corretas, enviar mensagens de erro caso o form não tenha sido preenchido corretamente, notificar advogados e atletas da situação do contrato e salvar as informações no banco. |
| Métodos | + verificaInformacoes() : boolean: Verifica se as informações preenchidas no form estão corretas  + criaContrato(codigo : string, numero : int, numeroLeis : int, taxa : int, compromisso : string, advogado : Advogado, atleta : Atleta) : void: Cria a entidade contrato com as informações passadas por parâmetro  + enviarMensagemErro(erro : string) : void: Envia uma mensagem de erro caso as informações preenchidas no form forem preenchidas de forma incorreta ou caso acontece algum outro erro  + alteraContrato(codigo : string, numero : int, numeroLeis : int, taxa : int, compromisso : string, advogado : Advogado, atleta : Atleta) : void: Altera as informações de um contrato de um atleta  + renovaContrato(atleta: Atleta) : Void: Renova o contrato de um atleta  + notifica(advogado : Advogado, atleta : Atleta) : void: Notifica o advogado e o atleta da situação atual do contrato  + salvar() : void: Salva as informações no banco |
| Atributos |  |

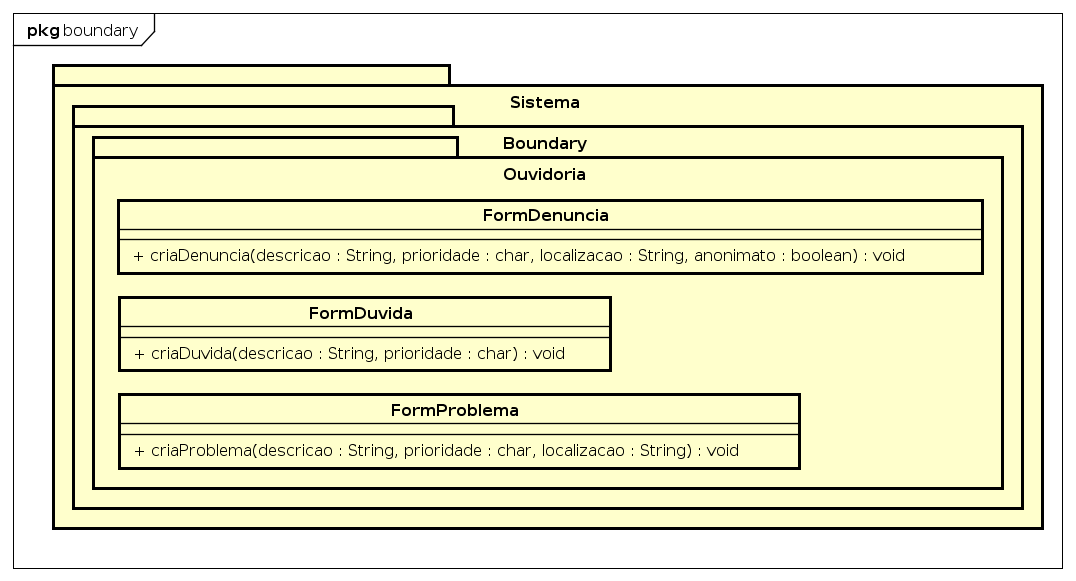
|  |  |
| --- | --- |
| **Classe EmailController** | |
| Descrição | Esta classe representa o controller que envia o email ao usuário que criou uma denúncia, problema ou dúvida |
| Relações | Possui relação com as boundaries FormDenuncia, FormProblema e FormDuvida |
| Responsabilidades | A principal responsabilidade desta classe é enviar o email para o usuário após a criação de uma Dúvida, Denúncia ou Problema |
| Métodos | + enviaEmail(usuario : int, descricao : String) : void |
| Atributos |  |

|  |  |
| --- | --- |
| **Classe PartidaController** | |
| Descrição | Esta classe representa o controller da partida e suas relações |
| Relações | Possui relação com a FormCompraIngresso e PagamentoController |
| Responsabilidades | A principal responsabilidade desta classe é criar a comunicação entre o FormCompraIngresso e FormPagamento |
| Métodos | + buscaPartidas(): Partida[] Retorna as partidas encontradas +verificaCadeiras(cadeiras: int): bool Verifica se as cadeiras selecionadas pelo cliente estão disponíveis. |
| Atributos |  |

|  |  |
| --- | --- |
| **Classe PagamentoController** |  |
| Descrição | Esta classe representa o controller do pagamento e suas relações |
| Relações | Possui relação com o PagamentoBoundary e PartidaController |
| Responsabilidades | A responsabilidade desta classe é ser o gateway com diversos serviços de validação de pagamento. |
| Métodos | validaPagamento(nome: String, numero: String, vencimento: int, cvv: int, bandeira: String ): boolean |
| Atributos |  |

### Pacote Boundary

#### Pacote Ouvidoria

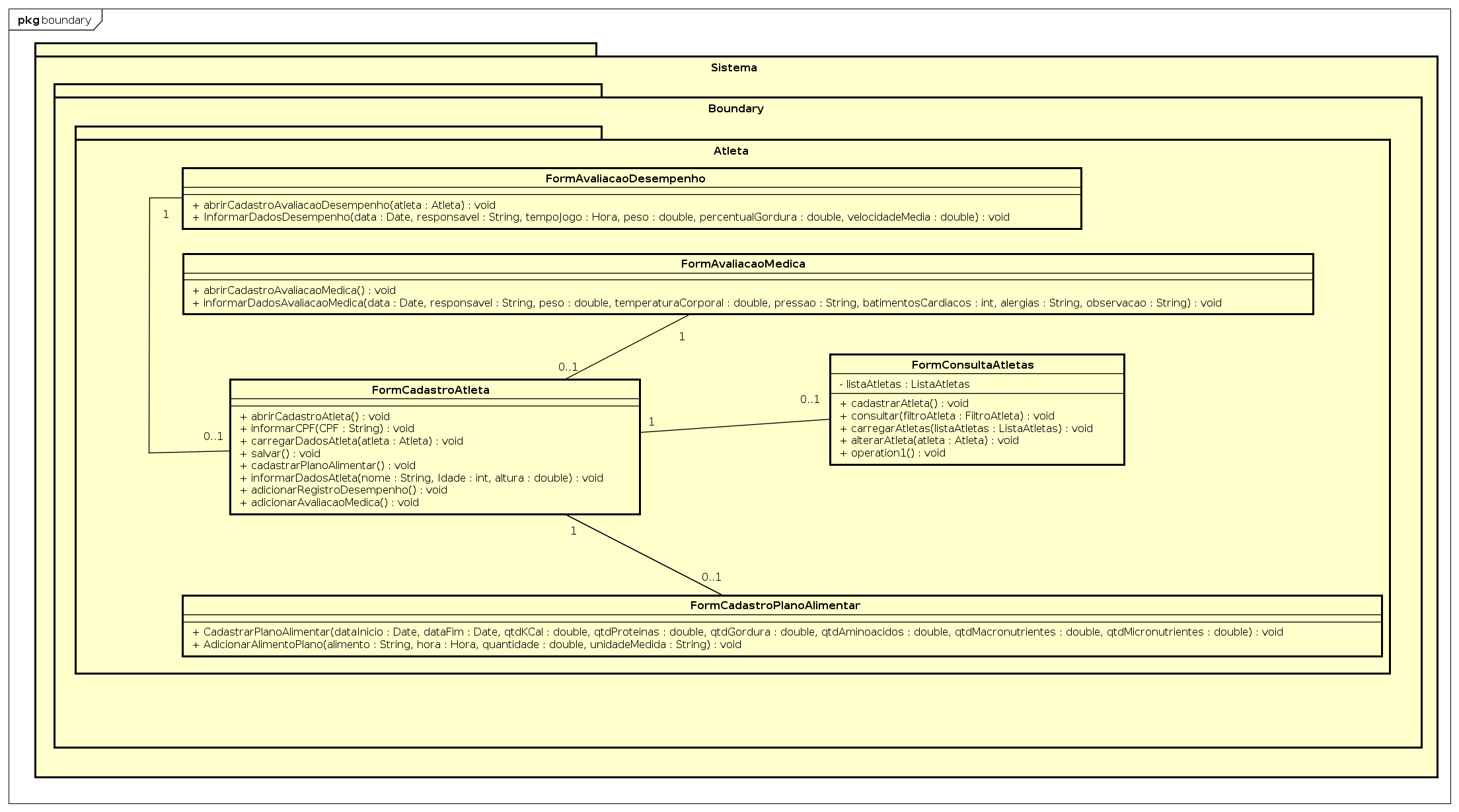


|  |  |
| --- | --- |
| Classe FormDenuncia |  |
| Descrição | Classe boundary responsável pelo preenchimento de informações para a abertura de um chamado do tipo denúncia |
| Relações | Possui relação com o controller EmailController e ChamadoController |
| Responsabilidades | A responsabilidade desta classe é passar as informações preenchidas pelo usuário ao respectivo controller |
| Métodos | + criaDenuncia(descricao : String, prioridade : char, localizacao : String, anonimato : boolean) : void |
| Atributos |  |

|  |  |
| --- | --- |
| Classe FormProblema |  |
| Descrição | Classe boundary responsável pelo preenchimento de informações para a abertura de um chamado do tipo problema |
| Relações | Possui relação com o controller EmailController e ChamadoController |
| Responsabilidades | A responsabilidade desta classe é passar as informações preenchidas pelo usuário ao respectivo controller |
| Métodos | + criaProblema(descricao : String, prioridade : char, localizacao : String) : void |
| Atributos |  |

|  |  |
| --- | --- |
| Classe FormDuvida |  |
| Descrição | Classe boundary responsável pelo preenchimento de informações para a abertura de um chamado do tipo duvida |
| Relações | Possui relação com o controller EmailController e ChamadoController |
| Responsabilidades | A responsabilidade desta classe é passar as informações preenchidas pelo usuário ao respectivo controller |
| Métodos | + criaDuvida(descricao : String, prioridade : char) : void |
| Atributos |  |

#### Pacote Atleta



Descrição das classes

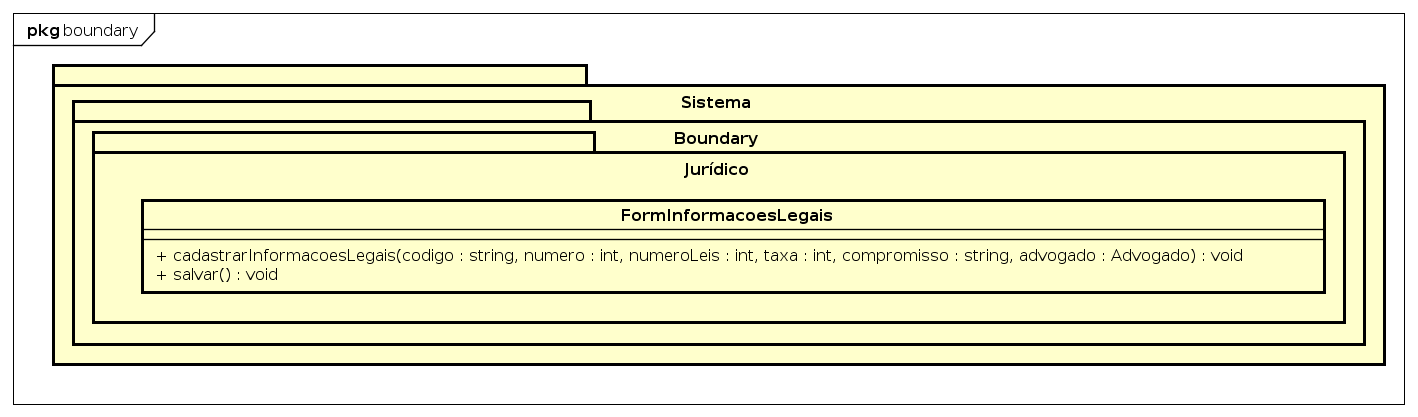
|  |  |
| --- | --- |
| Classe FormConsultaAtletas |  |
| Descrição | Classe boundary responsável pela consulta dos atletas do clube |
| Relações | Possui relação com as classes Atleta, AtletaController e ListaAtletas  Tem relação com o boundary FormCadastroAtleta |
| Responsabilidades | A responsabilidade desta classe é representar as consultas de Atletas, chamar alterações e cadastros de atletas |
| Métodos | + cadastrarAtleta() : void: chama o formulário de cadastro de atleta  + consultar(filtroAtleta : FiltroAtleta) : void  + carregarAtletas(listaAtletas : ListaAtletas) : void  + alterarAtleta(atleta : Atleta) : void: chama o formulário de cadastro de atleta para realizar alterações no cadastro |
| Atributos | listaAtletas |

|  |  |
| --- | --- |
| Classe FormCadastroPlanoAlimentar |  |
| Descrição | Classe boundary responsável pela manutenção de planos alimentares |
| Relações | Possui relação com as classes Atleta, AtletaController e ListaAtletas  Tem relação com o boundary FormCadastroAtleta |
| Responsabilidades | A responsabilidade desta classe é representar as consultas de Atletas, chamar alterações e cadastros de atletas |
| Métodos | + cadastrarAtleta() : void: chama o formulário de cadastro de atleta  + consultar(filtroAtleta : FiltroAtleta) : void  + carregarAtletas(listaAtletas : ListaAtletas) : void  + alterarAtleta(atleta : Atleta) : void: chama o formulário de cadastro de atleta para realizar alterações no cadastro |
| Atributos | listaAtletas |

|  |  |
| --- | --- |
| Classe FormAvaliacaoDesempenho |  |
| Descrição | Classe boundary que serve para a manutenção de avaliações de desempenho |
| Relações | Possui relação com as classes Atleta, AtletaController  Tem relação com o boundary FormCadastroAtleta |
| Responsabilidades | A responsabilidade desta classe é realizar a manutenção nos dados de avaliação de desempenho do atleta |
| Métodos | + abrirCadastroAvaliacaoDesempenho(atleta : Atleta) : void: abre o formulário de manutenção da avaliação de desempenhjo  + InformarDadosDesempenho(data : Date, responsavel : String, tempoJogo : Hora, peso : double, percentualGordura : double, velocidadeMedia : double) : void: chama o método de adicionar avaliação de desempenho ao objeto da entidade |
| Atributos |  |

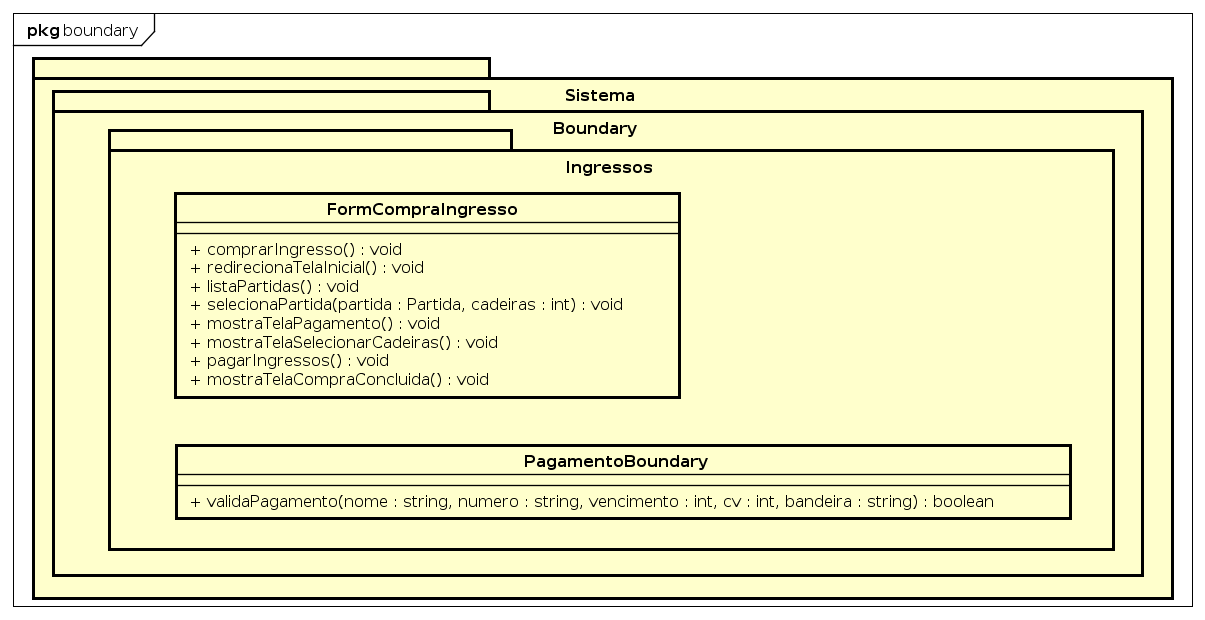
|  |  |
| --- | --- |
| Classe FormAvaliacaoMedica |  |
| Descrição | Classe boundary que serve para a manutenção de avaliações de desempenho |
| Relações | Possui relação com as classes Atleta, AtletaController  Tem relação com o boundary FormCadastroAtleta |
| Responsabilidades | A responsabilidade desta classe é realizar a manutenção nos dados de avaliação de desempenho do atleta |
| Métodos | + abrirCadastroAvaliacaoMedica() : void: abre o formulário de manutenção da avaliação médica  + informarDadosAvaliacaoMedica(data : Date, responsavel : String, peso : double, temperaturaCorporal : double, pressao : String, batimentosCardiacos : int, alergias : String, observacao : String) : void: chama o método de adicionar avaliação médica ao objeto da entidade |
| Atributos |  |

#### Pacote Jurídico



|  |  |
| --- | --- |
| Classe FormInformacoesLegais |  |
| Descrição | Classe boundary responsável pelo preenchimento de informações legais a respeito do contrato |
| Relações | Possui relação com o controller advogadoController |
| Responsabilidades | A responsabilidade desta classe é repassar as informações preenchidas pelo advogado no form para o respectivo controller |
| Métodos | + cadastrarInformacoesLegais(codigo : string, numero : int, numeroLeis : int, taxa : int, compromisso : string, advogado : Advogado) : void: cadastra as informações legais e as transfere para o controller advogadoController  + salvar(): void chama o método salvarInformacoesLegais do controller Advogado |
| Atributos |  |

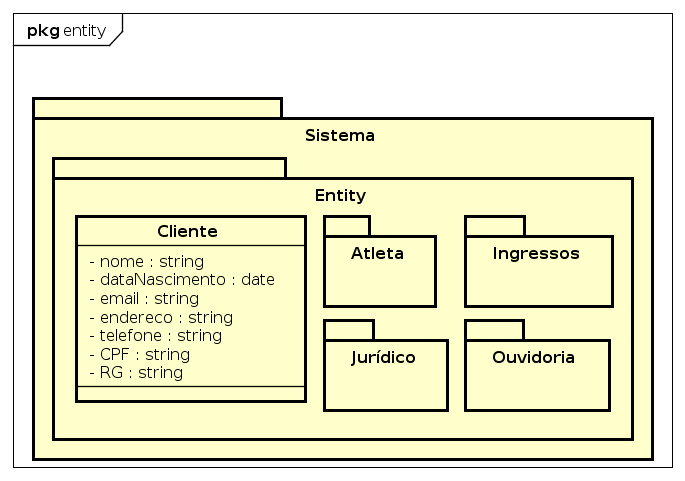
#### Pacote Ingressos



|  |  |
| --- | --- |
| Classe FormCompraIngresso |  |
| Descrição | Classe boundary responsável pelo venda online de ingressos. |
| Relações | Possui relação com os controller de pagamento. |
| Responsabilidades | A responsabilidade desta classe é interagir com o cliente durante a compra de ingresso online. |
| Métodos | + compraIngresso(): void Inicia o processo de aquisição de ingresso. + redirecionaTelaInicial(): void Processo que leva o cliente a tela inicial + listaPartidas(): void Lista as partidas disponiveis para o cliente + selecionaPartida(partida: Partida, cadeiras: int): void Cliente seleciona a partida que gostaria de assistir e em qual lugar. + mostraTelaPagamento(): void Mostra a tela de pagamento para o cliente  + mostraTelaSelecionarCadeiras(): void Mostra tela para seleção de cadeiras quando alguma cadeira selecionada previamente não está disponível  + pagarIngresso(): void Continua com o processo de compra de ingresso indo para o processo de validação + mostraTelaCompraConcluida(): void Exibe tela de finalização de compra |
| Atributos |  |

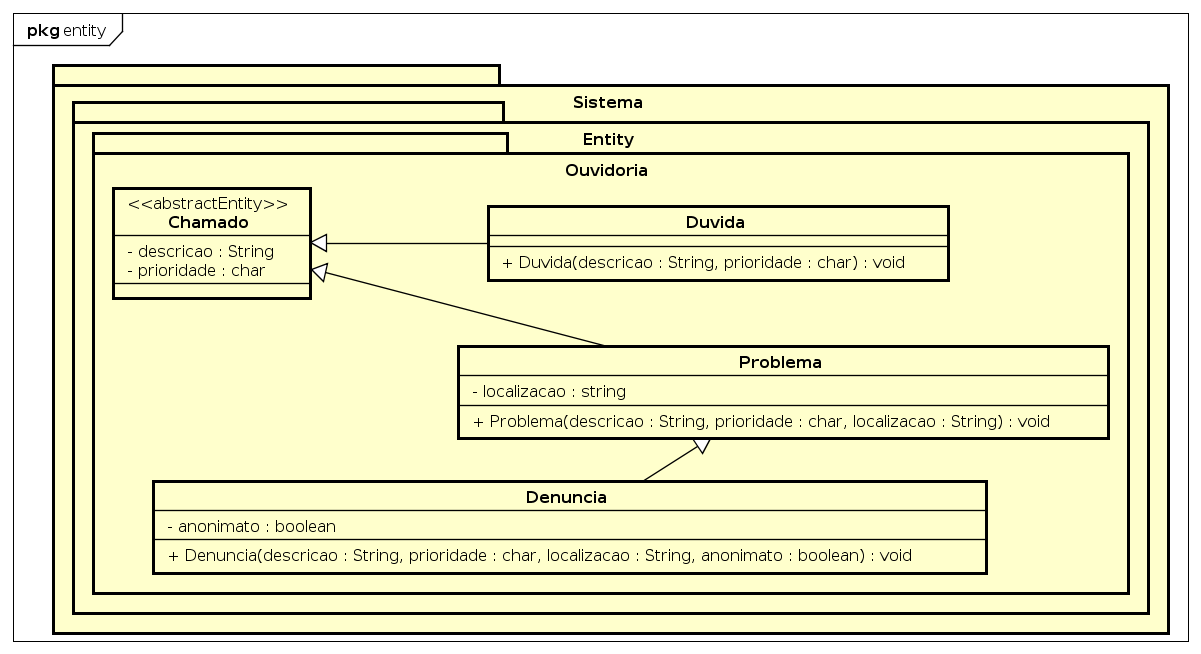
|  |  |
| --- | --- |
| Classe PagamentoBoundary |  |
| Descrição | Classe boundary responsável pela validação do pagamento com serviços externos. |
| Relações | Possui relação com os controller de pagamento. |
| Responsabilidades | A responsabilidade desta classe é ser o gateway com diversos serviços de validação de pagamento. |
| Métodos | validaPagamento(nome: String, numero: String, vencimento: int, cvv: int, bandeira: String ): boolean |
| Atributos |  |

### Pacote Entity



|  |  |
| --- | --- |
| Classe Cliente | |
| Descrição | Esta classe representa os dados do Cliente |
| Relações | Possui relação com as classes Problema, Duvida e Denuncia |
| Responsabilidades | A principal responsabilidade desta classe é representar o cliente e mantes seus dados |
| Métodos | +CriaProblema(cliente : Cliente, descricao : String) : void  +CriaDenuncia(cliente : Cliente, descricao : String) : void  +CriaDuvida(cliente : Cliente, descricao : String) : void |
| Atributos | nome  dataNascimento  email  endereco  telefone  cpf  rg |

#### Pacote Ouvidoria



Descrição das classes.

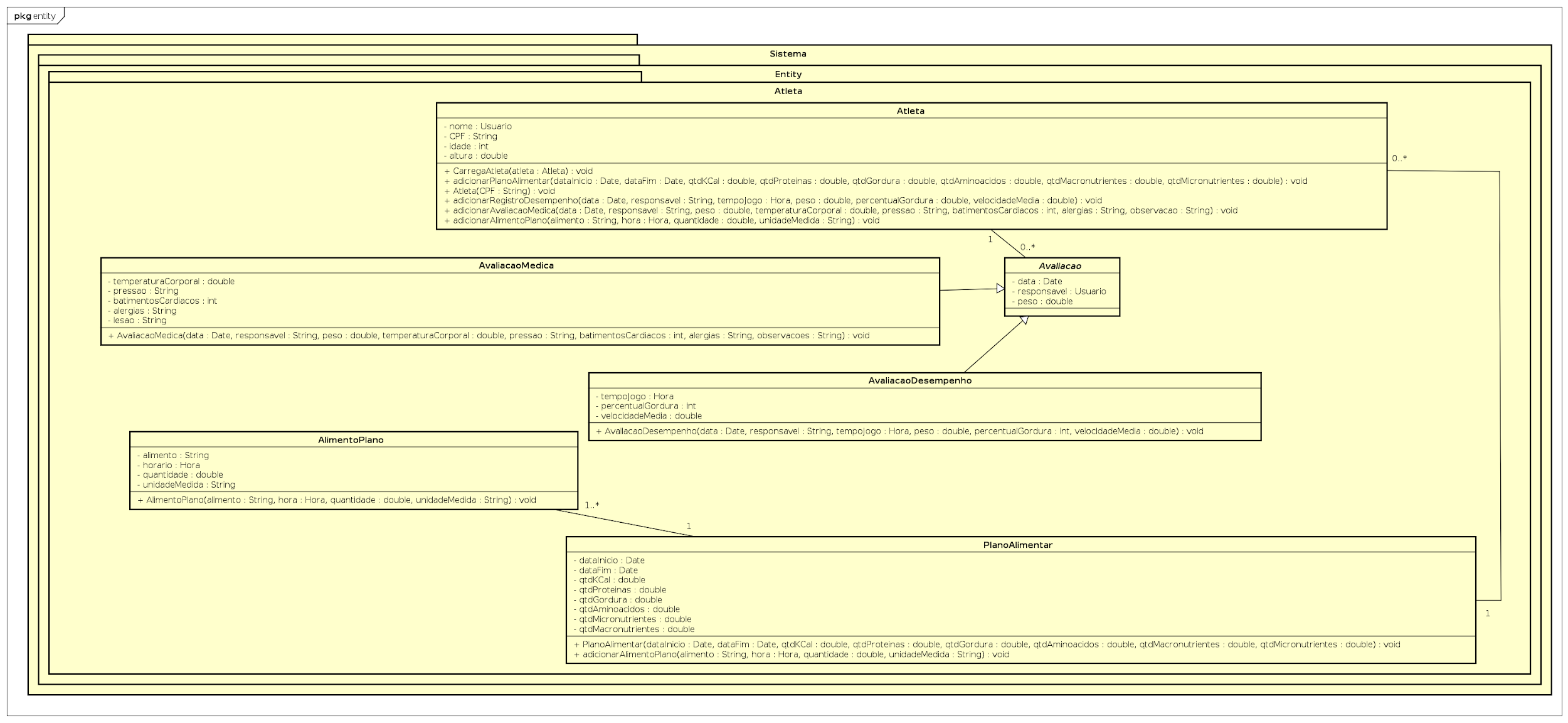
|  |  |
| --- | --- |
| Classe Chamado | |
| Descrição | Esta classe abstrata de dados de um chamado |
| Relações | Possui relação com a classe ChamadoController e é extendida pelas classes Problema, Denuncia e Duvida |
| Responsabilidades | A principal responsabilidade desta classe é generalizar os dados de um chamado |
| Métodos |  |
| Atributos | descricao: Descricão do problema encontrado  prioridade: Prioridade para resolução do caso |

|  |  |
| --- | --- |
| Classe Problema | |
| Descrição | Esta classe representa os dados apresentados em um problema |
| Relações | É filha da classe Chamado |
| Responsabilidades | A principal responsabilidade desta classe é apresentar os problemas criados pelos clientes |
| Métodos | + Problema(descricao : String, prioridade : char, localizacao : String) : void |
| Atributos | localizacao: Localização onde o problema foi encontrado |

|  |  |
| --- | --- |
| Classe Duvida | |
| Descrição | Esta classe representa os dados apresentados em uma dúvida |
| Relações | É filha da classe Chamado |
| Responsabilidades | A principal responsabilidade desta classe é apresentar as dúvidas criadas pelos clientes |
| Métodos | + Duvida(descricao : String, prioridade : char) : void |
| Atributos |  |

|  |  |
| --- | --- |
| Classe Denuncia | |
| Descrição | Esta classe representa os dados apresentados em uma denúncia |
| Relações | É filha da classe Chamado |
| Responsabilidades | A principal responsabilidade desta classe é apresentar as denúncias criadas pelos clientes |
| Métodos | + Denuncia(descricao : String, prioridade : char, localizacao : String, anonimato : boolean) : void |
| Atributos | localizacao: Localização onde algum ato aconteceu  anonimato: Cliente tem a possibilidade de esconder ou não sua identidade para fazer a denúncia |

#### Pacote Atleta



|  |  |
| --- | --- |
| Classe Atleta | |
| Descrição | Esta classe representa os dados do Atleta |
| Relações | possui associação com PlanoAlimentar e Avaliacao |
| Responsabilidades | A principal responsabilidade da classe de Atleta é representar um atleta e manter seus dados |
| Métodos | + Atleta(CPF : String) : void  + carregaAtleta(atleta: Atleta): recebe um objeto do tipo Atleta e copia os dados  + adicionarPlanoAlimentar(dataInicio : Date, dataFim : Date, qtdKCal : double, qtdProteinas : double, qtdGordura : double, qtdAminoacidos : double, qtdMacronutrientes : double, qtdMicronutrientes : double) : void  + adicionarRegistroDesempenho(data : Date, tempoJogo : Hora, peso : double, percentualGordura : double, velocidadeMedia : double) : void  + adicionarAvaliacaoMedica(data : Date, temperaturaCorporal : double, pressao : String, batimentosCardiacos : int, alergias : String, observacao : String) : void  + adicionarAlimentoPlano(alimento : String, hora : Hora, quantidade : double, unidadeMedida : String) : void |
| Atributos | nome  cpf  idade  altura |

|  |  |
| --- | --- |
| Classe Avaliacao |  |
| Descrição | Classe abstrata de dados de avaliação |
| Relações | é extendida por AvaliacaoDesempenho e Avaliacao e possui relação com a classe Atleta |
| Responsabilidades | A responsabilidade desta classe é generalizar os dados de Avaliacao dos atletas |
| Métodos |  |
| Atributos | data  responsavel  peso |

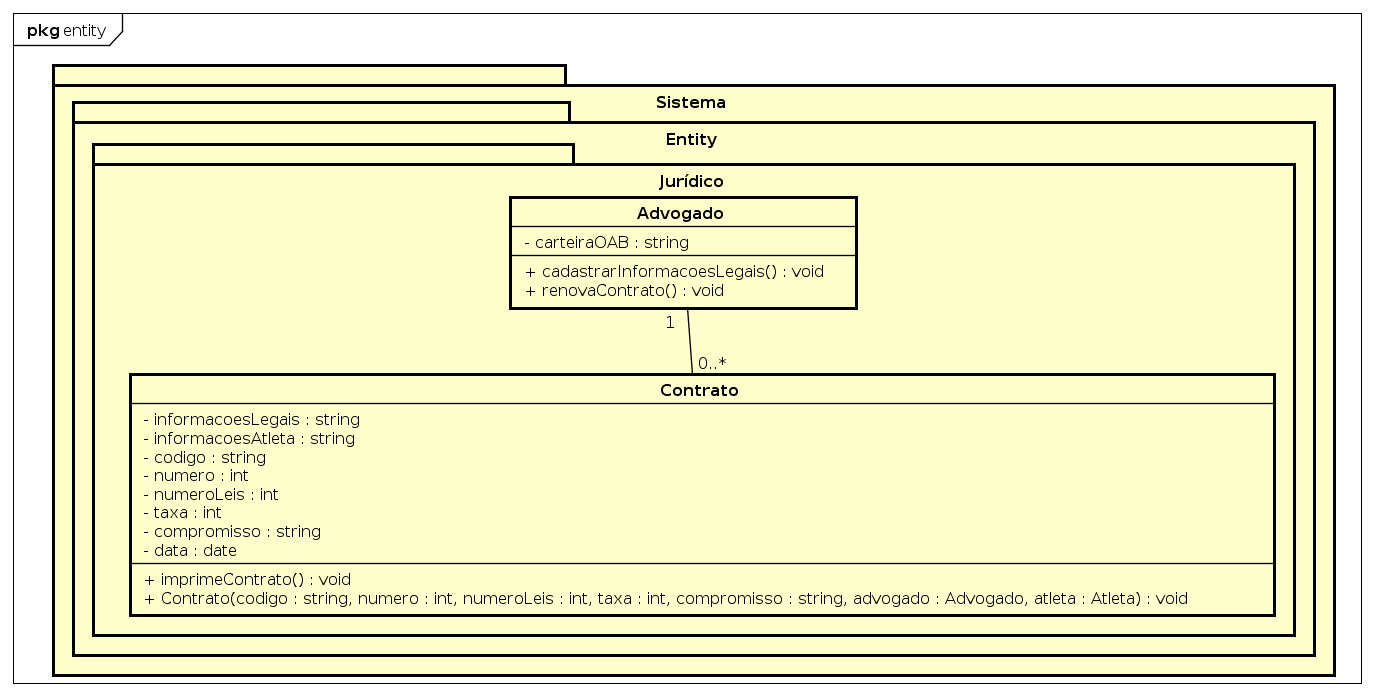
|  |  |
| --- | --- |
| Classe AvaliacaoDesempenho |  |
| Descrição | Classe de dados de Avaliacao de desempenho de atletas |
| Relações | extende a classe Avaliacao |
| Responsabilidades | A responsabilidade desta classe é armazenar os dados da AvaliacaoDesempenho |
| Métodos | + AvaliacaoDesempenho(data : Date, responsavel : String, tempoJogo : Hora, peso : double, percentualGordura : int, velocidadeMedia : double) : void: método de criação do objeto |
| Atributos | tempoJogo: tempo em que o jogador ficou em campo na partida  percentualGordura: percentual de gordura do atleta  velocidadeMedia: velocidade média de corrida do atleta |

|  |  |
| --- | --- |
| Classe AvaliacaoMedica |  |
| Descrição | Classe de dados de Avaliacao médica de atletas |
| Relações | extende a classe Avaliacao |
| Responsabilidades | A responsabilidade desta classe é armazenar os dados da AvaliacaoMedica |
| Métodos | + AvaliacaoMedica(data : Date, responsavel : String, peso : double, temperaturaCorporal : double, pressao : String, batimentosCardiacos : int, alergias : String, observacoes : String) : void: método de criação do objeto |
| Atributos | temperaturaCorporal: temperatura do atleta na avaliação  pressao: pressão sanguínea do atleta na avaliação  batimentosCardiacos  alergias: lista de alergias do atleta  lesao: campo preenchido quando há lesões no atleta |

|  |  |
| --- | --- |
| Classe PlanoAlimentar |  |
| Descrição | Classe de dados de Planos Alimentares de atletas |
| Relações | possui relação com a classe Atleta e com a classe AlimentoPlano |
| Responsabilidades | A responsabilidade desta classe é armazenar os dados da PlanoAlimentar |
| Métodos | + PlanoAlimentar(dataInicio : Date, dataFim : Date, qtdKCal : double, qtdProteinas : double, qtdGordura : double, qtdAminoacidos : double, qtdMacronutrientes : double, qtdMicronutrientes : double) : void  + adicionarAlimentoPlano(alimento : String, hora : Hora, quantidade : double, unidadeMedida : String) : void |
| Atributos | dataInicio: data do início da validade do plano alimentar  dataFim: data do fim da validade do plano alimentar  qtdKcal: quantidade total de quilo calorias  qtdProteinas: quantidade total de proteínas  qtdGordura: quantidade total de gorduras do plano  qtdAminoacidos: quantidade total de aminoacidos do plano  qtdMicronutrientes: quantidade total de micronutrientes do plano  qtdMacronutrientes: quantidade total de Macronutrientes do plano |

|  |  |
| --- | --- |
| Classe AlimentoPlano |  |
| Descrição | Classe de dados de Alimento Plano de atletas |
| Relações | possui relação com a classe PlanoAlimentar |
| Responsabilidades | A responsabilidade desta classe é armazenar os dados da AlimentoPlano |
| Métodos | + AlimentoPlano(alimento : String, hora : Hora, quantidade : double, unidadeMedida : String) : void |
| Atributos | alimento  horario  quantidade  unidadeMedida |

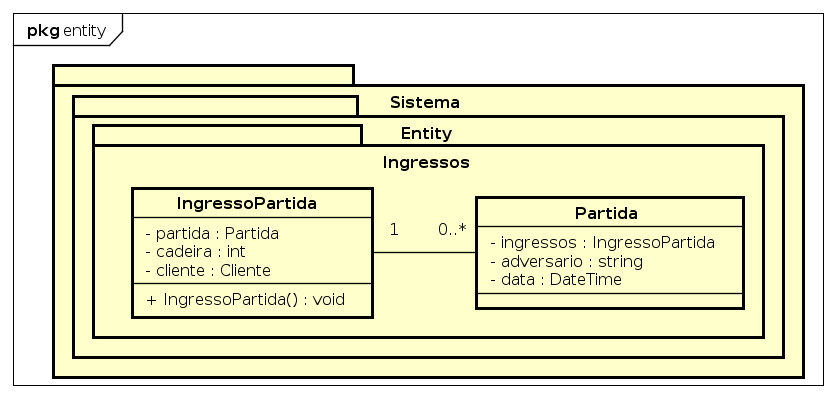
#### Pacote Jurídico



|  |  |
| --- | --- |
| Classe Contrato | |
| Descrição | Esta classe representa o contrato do Atleta |
| Relações | Possui relação com o controller contratoController |
| Responsabilidades | A principal responsabilidade desta classe é representar o contrato do atleta e manter seus dados |
| Métodos | + imprimeContrato() : void: Imprime as informações do contrato |
| Atributos | informacoesLegais  informacoesAtleta  codigo  numero  numeroLeis  taxa  compromisso  advogado  atleta  data |

|  |  |
| --- | --- |
| Classe Advogado | |
| Descrição | Esta classe representa o advogado que irá preencher o contrato do Atleta |
| Relações | Possui relação com o controller advogadoController |
| Responsabilidades | A principal responsabilidade desta classe é a de realizar ações relacionadas às informações legais do contrato do atleta |
| Métodos | + cadastrarInformacoesLegais() : void: Cadastra as informações legais do contrato do atleta  + renovaContrato() : void: Renova o contrato do atleta |
| Atributos | informacoesLegais  informacoesAtleta  codigo  numero  numeroLeis  taxa  compromisso  advogado  atleta  data |

#### Pacote Ingressos

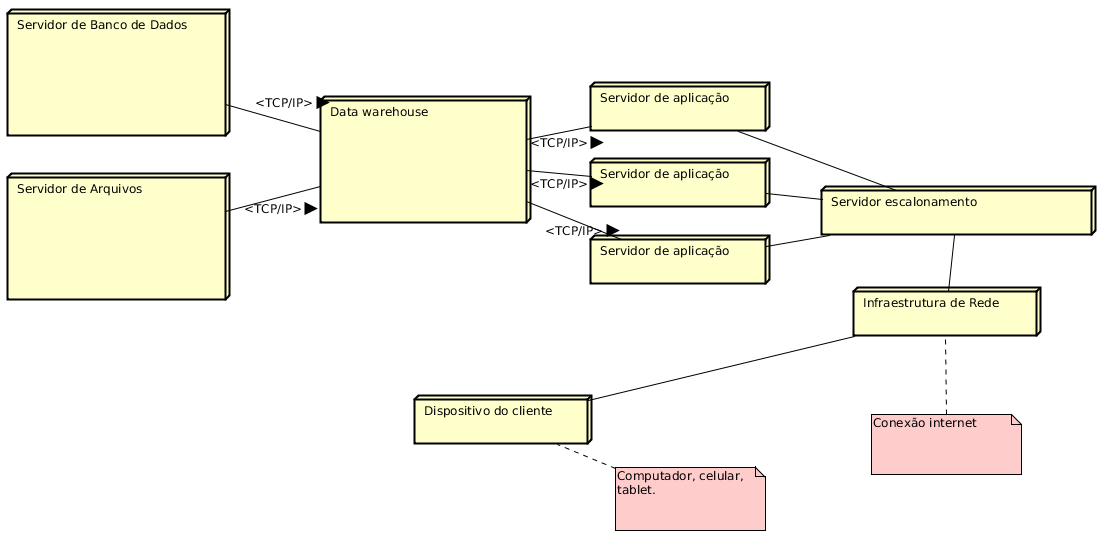


|  |  |
| --- | --- |
| Classe Partida | |
| Descrição | Esta classe representa os dados de uma Partida |
| Relações | Possui relação com a class IngressoPartida |
| Responsabilidades | Representar os dados de um Partida |
| Métodos |  |
| Atributos | Ingressos: Lista com os ingressos associados a uma partida  adversario: Nome do adversário para a partida  data: Data em que o jogo irá ocorrer. |

|  |  |
| --- | --- |
| **Classe IngressoPartida** | |
| Descrição | Classe que representa os dados de um ingresso para uma partida. |
| Relações | Classe dependente da partida e que se relaciona com o cliente |
| Responsabilidades | Tem como responsabilidade a confirmação da aquisição do ingresso para a partida de um cliente. |
| Métodos |  |
| Atributos | partida: Partida a qual o ingresso está associado. cadeira: Número da cadeira que o cliente comprou  cliente: Cliente responsável de aquisição do ingresso. |

# Visão de Implantação

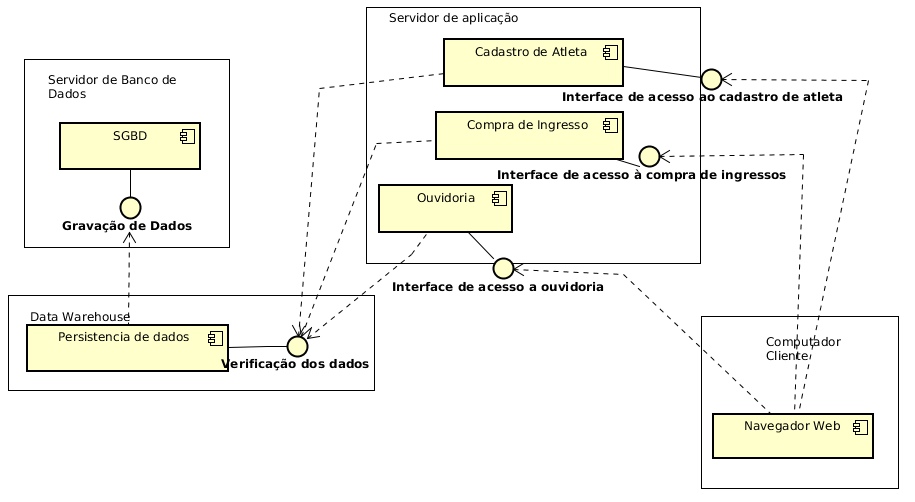
Essa implantação será composta de dois servidores, sendo um deles o banco de dados e o outro composto de arquivos. Essa informação será transmitida através de uma conexão TCP/IP para uma *data warehouse* que, por sua vez, irá transmitir essas informações para servidores da aplicação. As requisições serão manejadas pelo servidor de escalonamento que irá escalonar as requisições para cada servidor.



# Visão de Implementação

A visão de implementação aborda a arquitetura sobre a perspectiva do projeto estrutural dos componentes do sistema, como o sistema e cada um dos seus componentes serão organizados em termos de diretórios e como o sistema, será empacotado para publicação (deployment).

A visão de implementação retrata a composição física da implementação em termos de subsistemas e elementos (diretórios, arquivos, dados, executáveis). Geralmente essas camadas de implementação se ajustam às camadas definidas na visão lógica.



## Camadas

Esta subseção nomeia e define as diversas camadas e o seu conteúdo, as regras que determinam a inclusão em uma camada específica e as fronteiras entre as camadas.

* + 1. Camada de Apresentação

Para a camada de apresentação para o usuário utilizaremos uma página web baseada em HTML, CSS e JavaScript. O conteúdo View será renderizado através do React.

* + 1. Camada de Controle

A camada de controle será responsável pelo recebimento de arquivos JSON que deverão estar encriptados em Base64. Estes arquivos serão recebidos por chamadas REST, implementados em um servidor web em NodeJS.

* + 1. Camada de Domínio

A camada de domínio será responsável pelas operações de negócios e armazenamento dos dados no banco de dados e no servidor de arquivos.

## Soluções utilizadas

* + 1. HTML

HTML é uma linguagem de marcação de texto que ajuda na estruturação da página. O objetivo da utilização dessa linguagem é para que exista uma estrutura na página.

* + 1. CSS

CSS é a linguagem utilizada para a estilização da página. O objetivo da sua utilização é para que a página seja mais user-friendly. Os benefícios da sua utilização são os de melhor usabilidade da página.

* + 1. React

React é uma biblioteca de JavaScript que será utilizada com o objetivo de renderizar as páginas HTML. Seu benefício é possuir uma maior manutenibilidade do código e maior reutilização de código.

* + 1. JSON

JSON é utilizado como um objeto de notação. O modelo JSON é utilizado para armazenar os arquivos recebidos pelo back-end. Um dos benefícios de utilizar JSON ao invés de XML é o fato de que o JSON é auto-descritivo.

* + 1. NodeJS

O NodeJS é uma ferramenta utilizada para que seja possível o desenvolvimento de aplicações backend usando a linguagem Javascript, sem que elas precisem de um navegador *web* para executarem.

* + 1. Angular

É uma *framework* para desenvolvimento de aplicações web baseada em TypeScript, desenvolvida e mantida pela equipe da Google.

* + 1. TypeScript

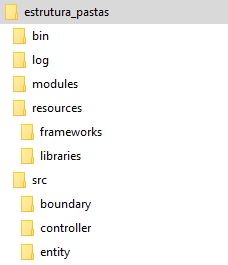
Typescript é uma linguagem criada pela Microsoft que é transpilada para JavaScript e seu objetivo é permitir tipação de dados.

* + 1. JavaScript

A linguagem JavaScript é utilizada como o controle das ações realizadas nas páginas. O objetivo de utilizar essa linguagem é para possuir uma vasta possibilidade de bibliotecas que possam ser utilizadas, como o React, por exemplo. O benefício do JavaScript é que ele é altamente adaptável e possui diversas bibliotecas que podem se encaixar para cada contexto.

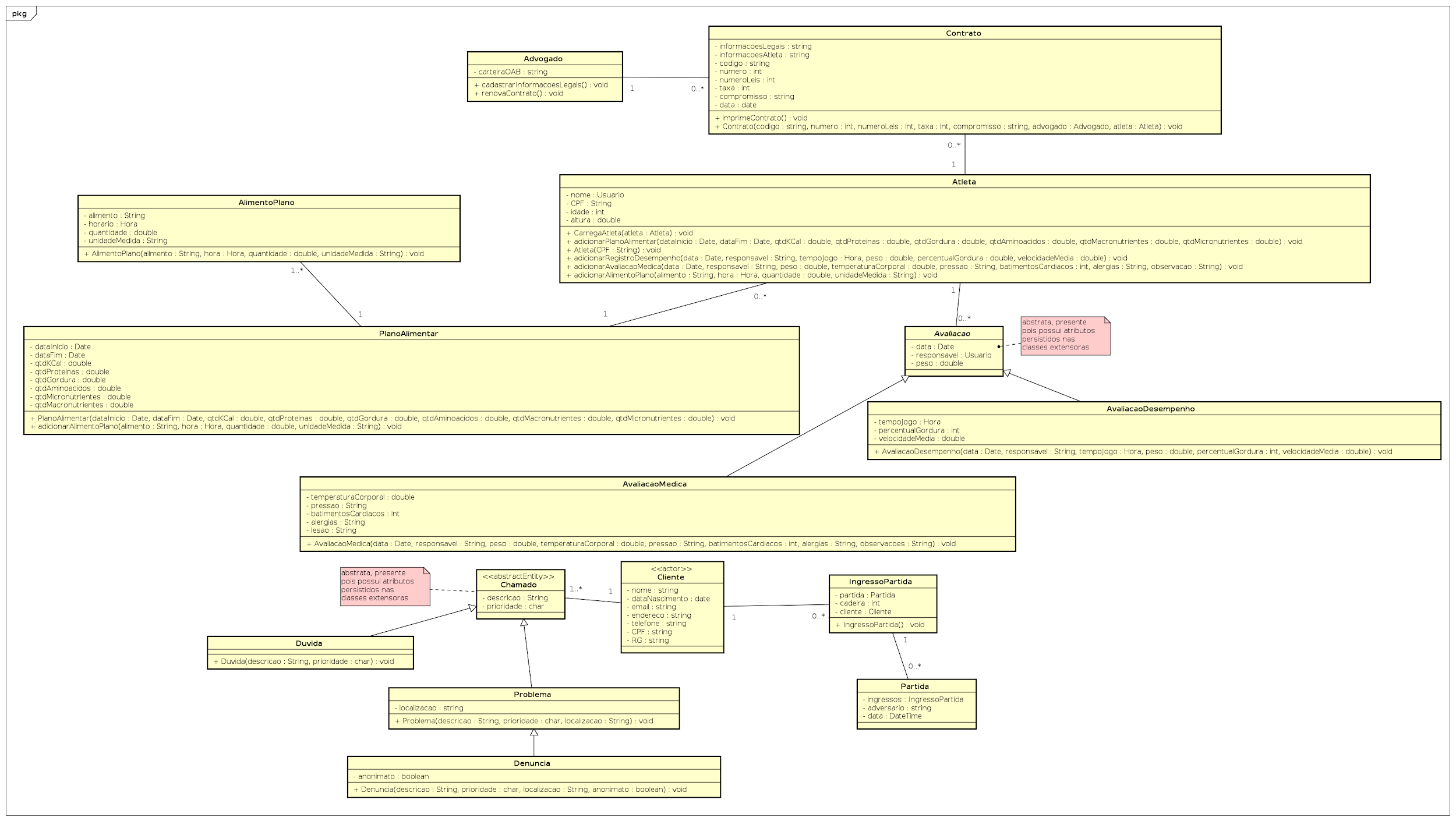
## Convenções

* + 1. Nomes de classes
* Classes de entidade não necessitam nenhum prefixo e sufixo;
* Classes de formulários de usuário necessitarão do sufixo “Form”;
* Classes de controlador necessitarão do sufixo “Controller”;
  + 1. Estrutura de diretórios



# Visão de Dados

## Modelo de objetos persistentes



## Estratégias

* O mapeamento utilizado pouco difere do modelo de design, uma vez que este foi feito considerando uma modelagem orientada a documentos, onde se utiliza o padrão JSON. A ferramenta de SGBD a ser utilizada é a MongoDB, por sua vasta documentação, fácil configuração e simplicidade de manutenção e mapeamento dos objetos.

## Modelo relacional

