

Documento de Arquitetura de Software

Concurso de Ideias de Negócio - CIN

Histórico da Revisão

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versão** | **Descrição** | **Autor** |
| 31/05/2015 | 0.1 | Elaboração do documento | Wellington Lucas Moura |
| 29/12/2015 | 0.1 | Adição da visão de implementação | Wellington Lucas Moura |
| 30/12/2015 | 0.1 | Acréscimo de novos diagramas na visão de Processos e Implementação | Wellington Lucas Moura |
|  |  |  |  |

1. **Introdução**
   1. **Finalidade**

Este documento apresenta a arquitetura proposta para a solução web de apoio ao jogo Concurso de Ideias de Negócio (CIN), utilizado na disciplina de Empreendedorismo na UFC – Campus Quixadá. O objetivo deste documento é capturar e comunicar as decisões arquiteturais significativas que foram tomadas em relação ao sistema.

* 1. **Escopo**

Este documento trata da arquitetura da solução web para o jogo Concurso de Ideias de Negócio, que visa apoiar a execução do jogo na disciplina de Empreendedorismo na UFC campus Quixadá.

* 1. **Definições, Acrônimos e Abreviações**

|  |  |
| --- | --- |
| Universidade Federal do Ceará | UFC |
| Requisito Arquitetural | RA |
| Model View Control | MVC |
| Spring Framework | SF |
| Núcleo de Práticas em Informática | NPI |
| Concurso de Ideias de Negócio | CIN |
| Unified Modeling Language | UML |

* 1. **Visão Geral**

Este documento está organizado em tópicos relacionados às diferentes visões arquiteturais abordadas.

1. **Arquitetura da Aplicação**

Esse documento apresenta a arquitetura como uma série de visualizações/visões: visão lógica, visão de processos e visão de execução. Essas visualizações são apresentadas utilizando-se UML.

* 1. **Representação arquitetural**

O sistema está sendo desenvolvido tendo como base a arquitetura adotada no NPI. Onde utiliza-se uma Arquitetura de Camadas juntamente com o Padrão Repository.

* 1. **Objetivos e Restrições da Arquitetura**

A arquitetura proposta tem como objetivo separar os interesses de cada camada da aplicação, disponibilizar um sistema funcional e com potencial de adaptabilidade a mudanças.

* 1. **Critérios da Avaliação Arquitetural**

Os critérios utilizados para a seleção da solução arquitetural foram:

* Modularidade;
* Modificabilidade;
* Manutenibilidade;

1. **Metas e Restrições da Arquitetura**

Softwares Utilizados

Para a execução do sistema será requisitado a instalação do (s) seguinte (s) software (s):

* Mozilla Firefox v. 38 ou superior / Google Chrome v.43 ou superior;

Caso algum software indicado não seja instalado, a aplicação pode não funcionar como esperado, ou não funcionar.

Existem alguns importantes requisitos e restrições do sistema que possuem uma influência significativa na arquitetura: São elas:

* Todas as funções devem estar disponíveis através dos navegadores mais populares;
* O sistema deve garantir a confidencialidade dos dados cadastrados;
* O sistema deve possuir diferentes níveis hierárquicos de acesso.

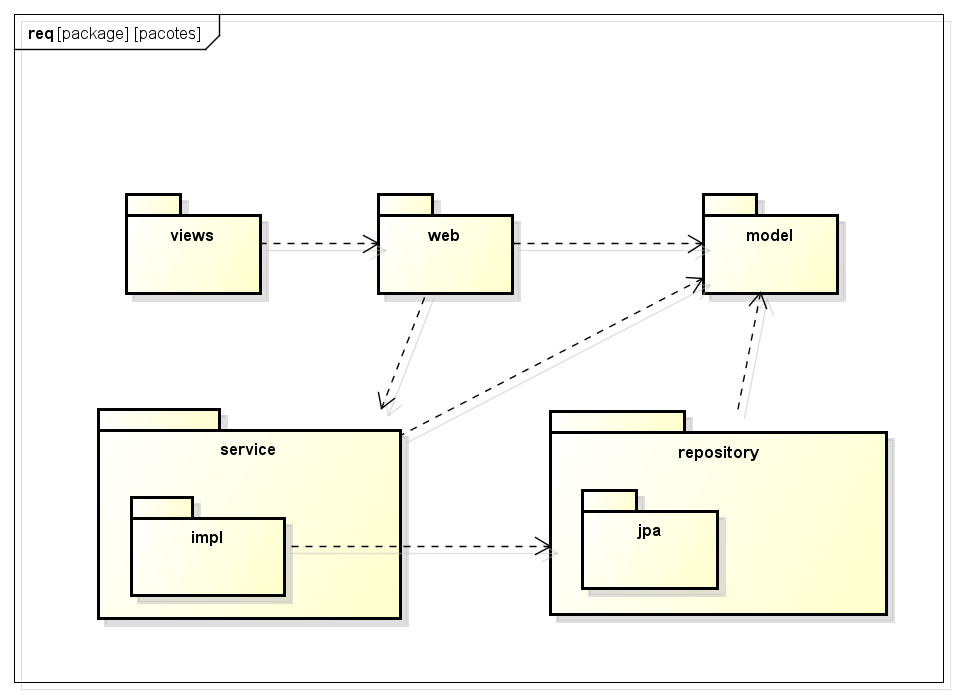
1. **Visualização Lógica**

Esta seção apresenta diferentes tipos de visões lógicas, com níveis de abstrações distintas, para que se possa obter uma visibilidade de comunicação entre pacotes e classes.

* 1. **Visão de pacotes**

A visão lógica da solução CIN é composta por 7 diretórios principais:

Figura 1 – Visão Lógica de Pacotes



Fonte: Produzido pelo autor

* **views**

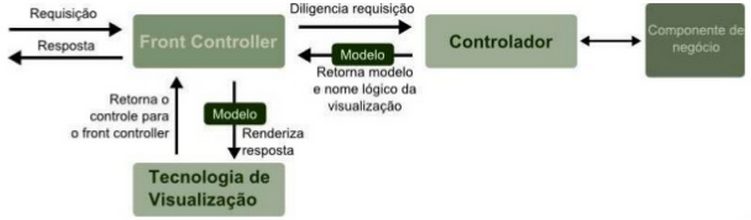
Neste diretório estão todos os subdiretórios e arquivos “.jsp” necessários para gerar a visualização do sistema. Aqui serão inseridos qualquer novo arquivo “.jsp” necessário para a visualização de informações.

* **web**

Neste diretório são implementados os controladores, responsáveis por fazer um “link” entre as regras de negócio da aplicação com a visão. SF usa o padrão Front Controller. O Front Controller intercepta todas as requisições para direcionar ao controlador adequado. A figura 2 apresenta o padrão Front Controller.

É importante que qualquer novo controlador seja inserido neste diretório para manter o padrão adotado de separação de interesses. Essa é uma decisão de projeto adotada para facilitar futuras manutenções de funcionalidades.

Figura 2 – Padrão Front Controller



Fonte: Retirado de (WEISSMANN, 2013)

* **model**

Contém as classes de negócio da aplicação. Ex.: Aluno. Qualquer nova entidade mapeada para o projeto deve ser inserida neste pacote.

* **repository**

Contém as interfaces de persistência das entidades do sistema. Ex.: AlunoRepository. Novas interfaces de repositório devem ser inseridos nesse diretório.

* **repository.jpa**

Contém as classes que implementam as interfaces de repository. Ex.: JpaAlunoRepository. Ao ser inserida alguma interface no pacote repository deve-se incluir aqui uma implementação para ela.

* **service**

Contém as interfaces responsáveis pela lógica relacionada às entidades do sistema. Ex.: AlunoService. Novas interfaces de serviço devem ser incluídas nesse pacote, à medida que se faça necessário.

* **service.impl**

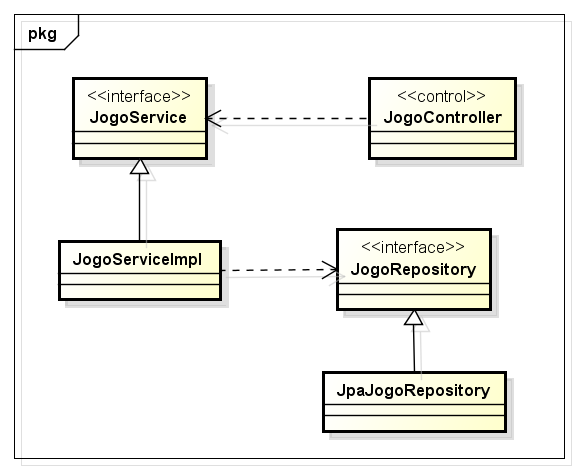
Contém as classes que implementam as interfaces de service. Ex.: AlunoServiceImpl.

* 1. **Visão de Classes**

Apresenta como as classes se comunicam entre si na aplicação.

* + 1. **Interação entre classes dos pacotes repository, service e web**

Figura 3 – Padrão de comunicação entre classes e interfaces

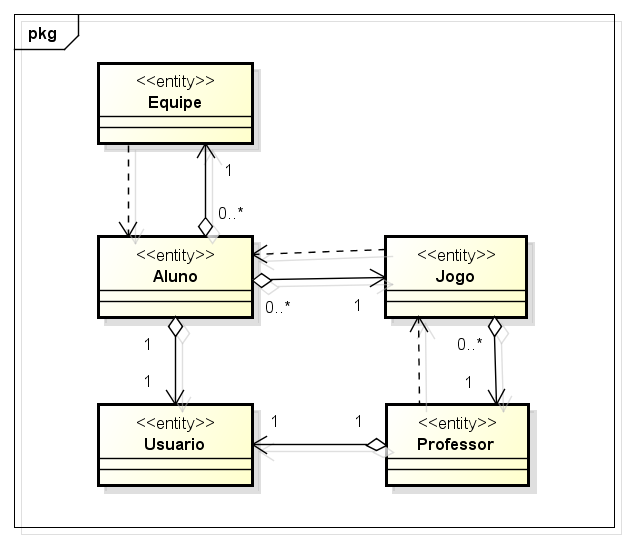


Fonte: Produzido pelo autor

As classes se relacionam dessa forma para aumentar a abstração dos dados, sendo necessário que uma classe, que precise de um serviço, conheça apenas as interfaces que fornecem o serviço.

* + 1. **Dependências entre classes do modelo**

Figura 4 – Entidades do pacote model



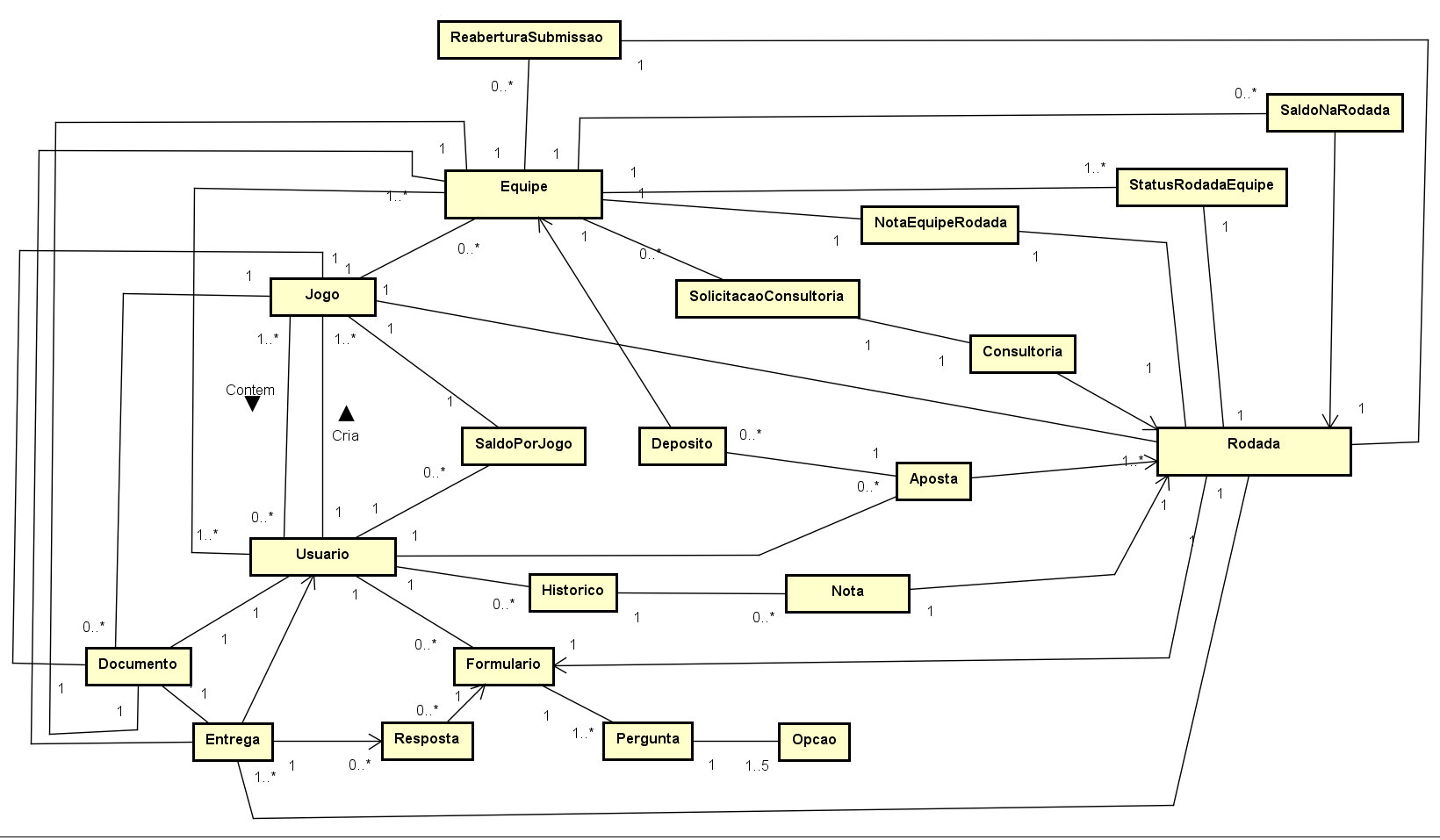
Fonte: Produzido pelo autor

Mostra as dependências não estruturais e as agregações entre as classes, com suas respectivas cardinalidades entre elas.

* + 1. **Divisão de classes**

Mostra a visão geral de como as classes do sistema se comunicam.

Figura 6 – Visão das classes de modelo

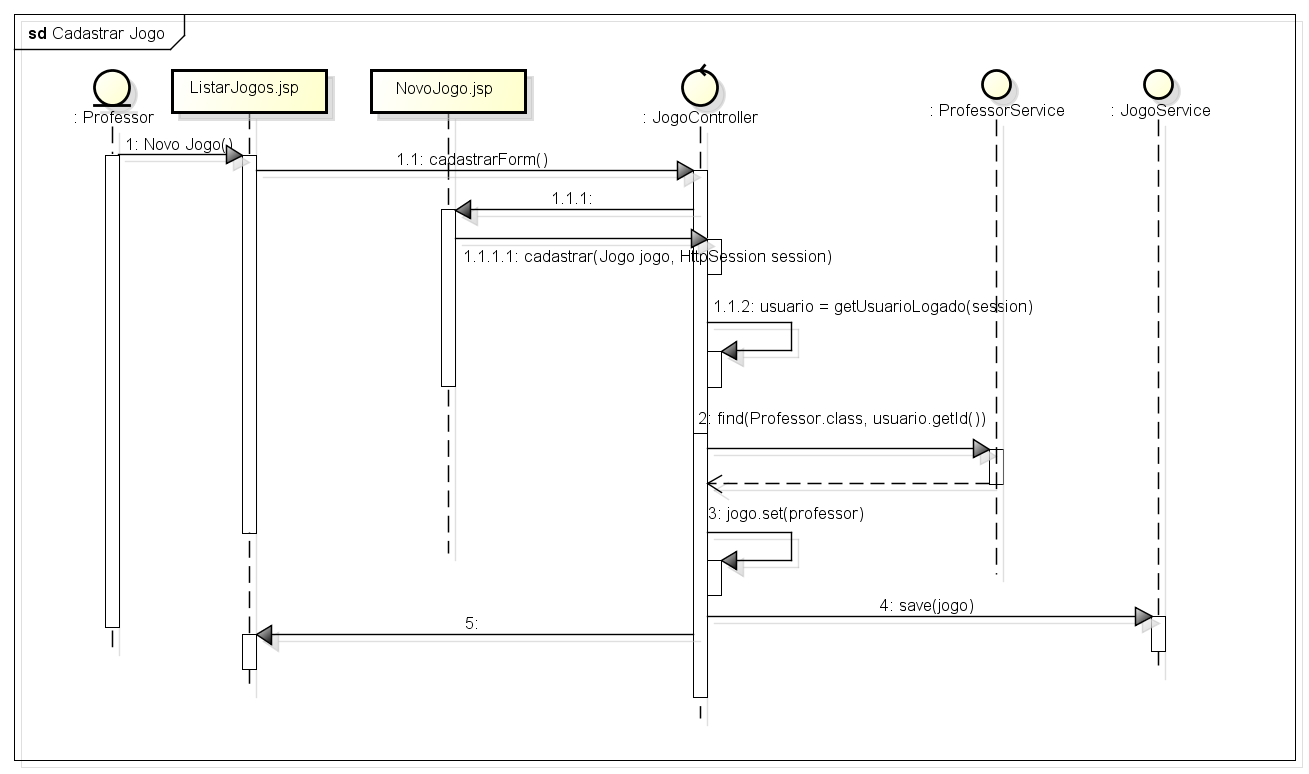
****

Fonte: Produzido pelo autor

1. **Visão de Processos**

Esta seção descreve a decomposição do sistema em processos, sendo evidenciado até então, o processo de cadastrar um novo jogo no sistema. No qual um professor logado no sistema faz uma requisição para se criar um novo jogo, o fluxo é apresentado na Figura 17.

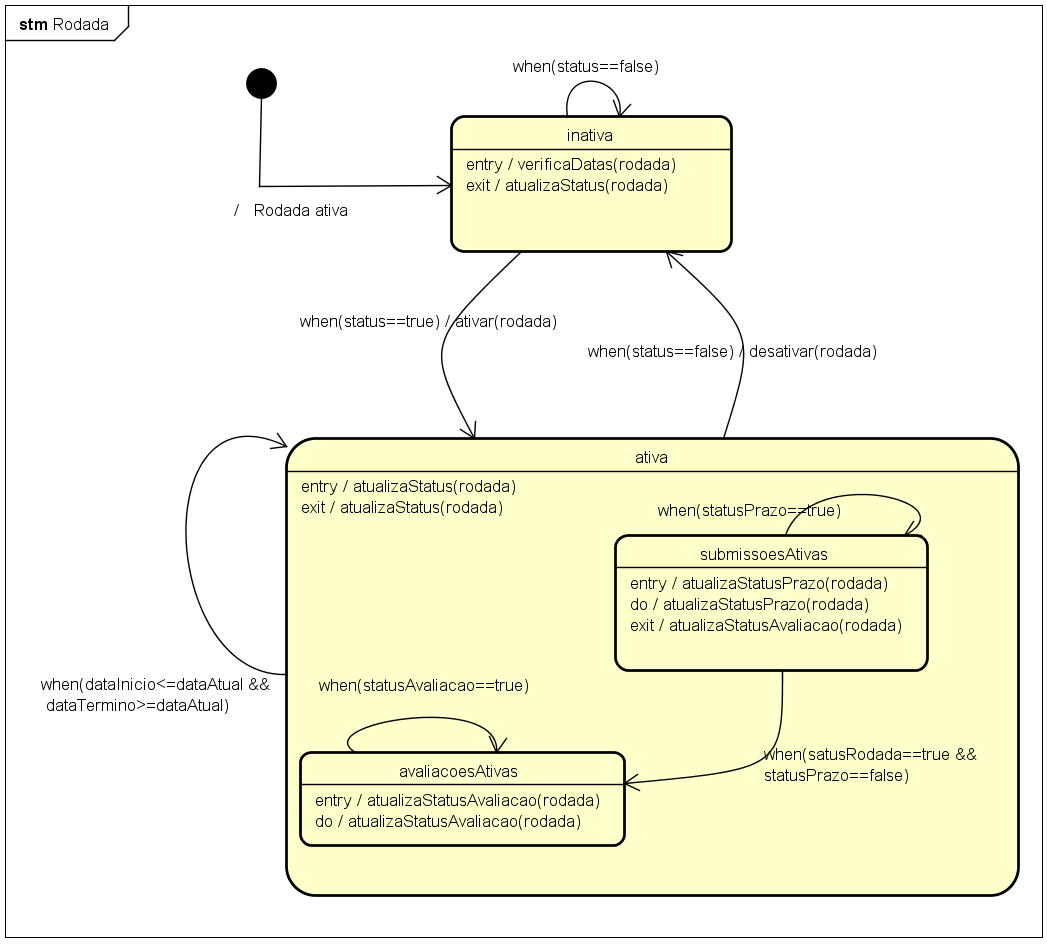
Figura 7 – Cadastrar novo jogo



Fonte: Produzido pelo autor

A seguir é apresentado um diagrama de máquina de estados de uma rodada, considerando-se que é a entidade que possui mais estados no jogo e a maior parte da lógica do sistema ocorre neste modelo.

Figura 8 – Diagrama de estado de máquina de uma rodada



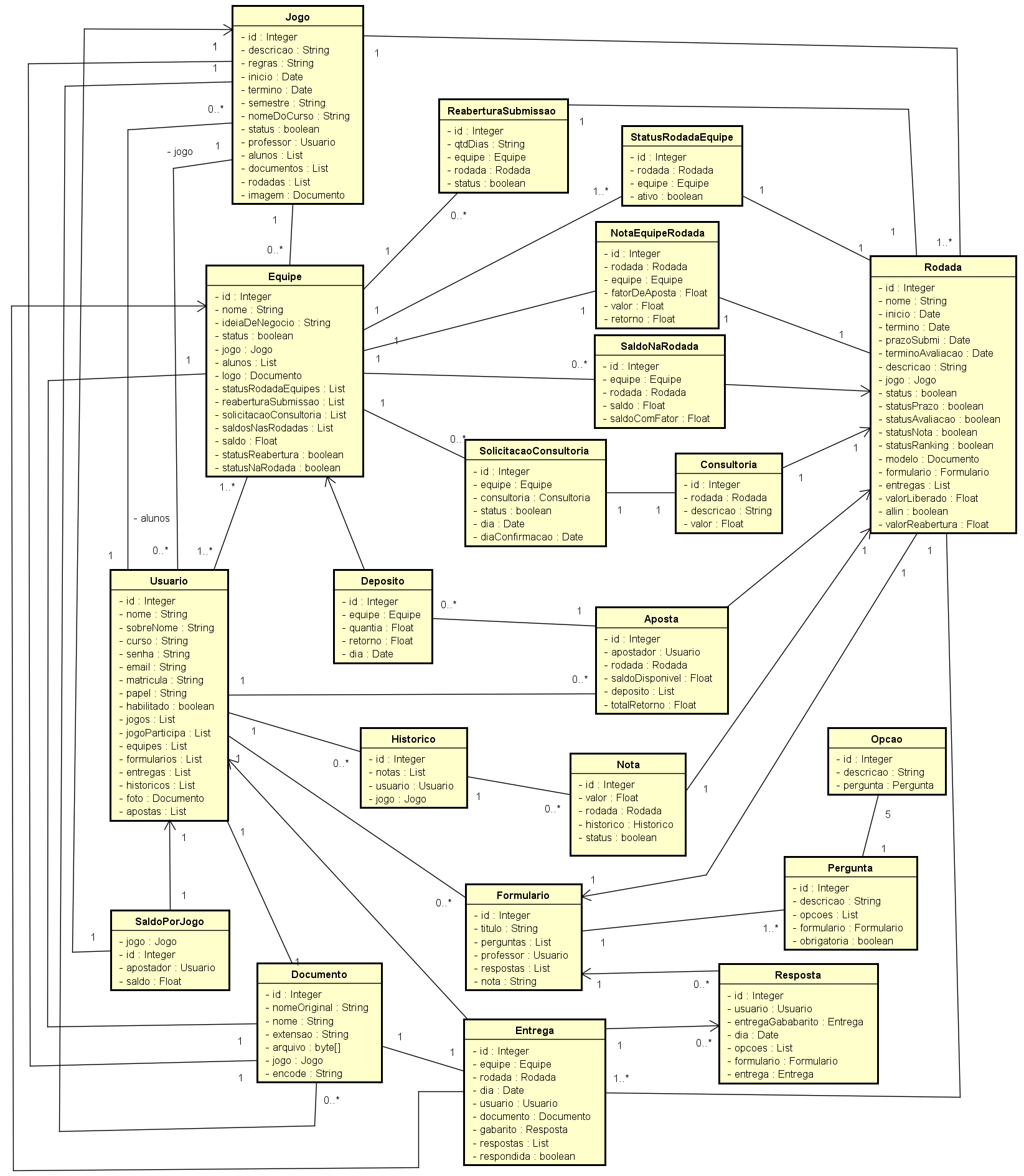
Fonte: Produzido pelo autor

1. **Visão de Implementação**

Esta seção descreve a estrutura geral do modelo de implementação, a divisão do software em subsistemas no modelo de implementação e todos os componentes significativos do ponto de vista da arquitetura.

* 1. **Visão geral das classes de modelo**

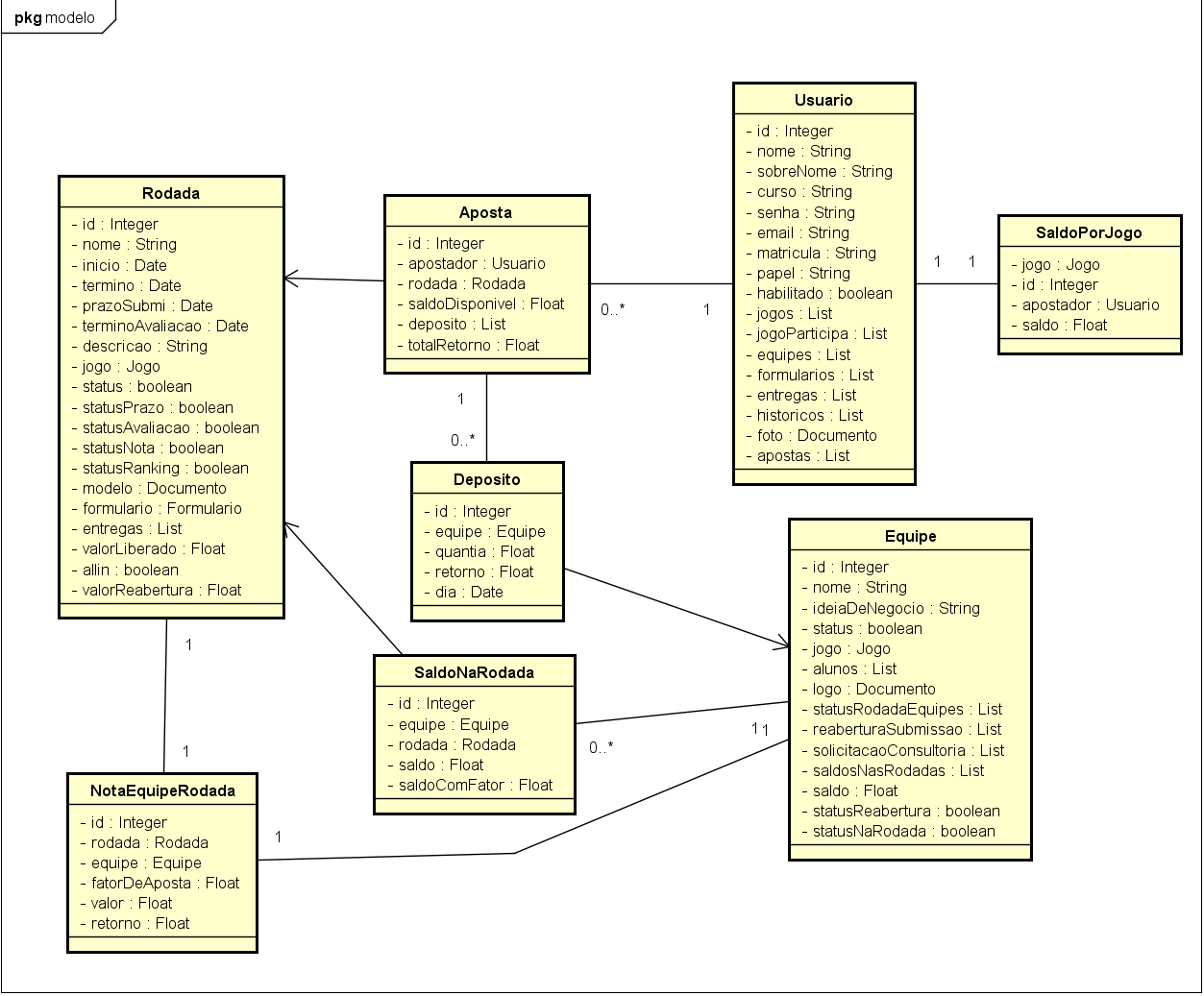
Figura 9 – Visão de classes de modelo



Fonte: Produzido pelo autor

* 1. **Visão de apostas numa rodada**

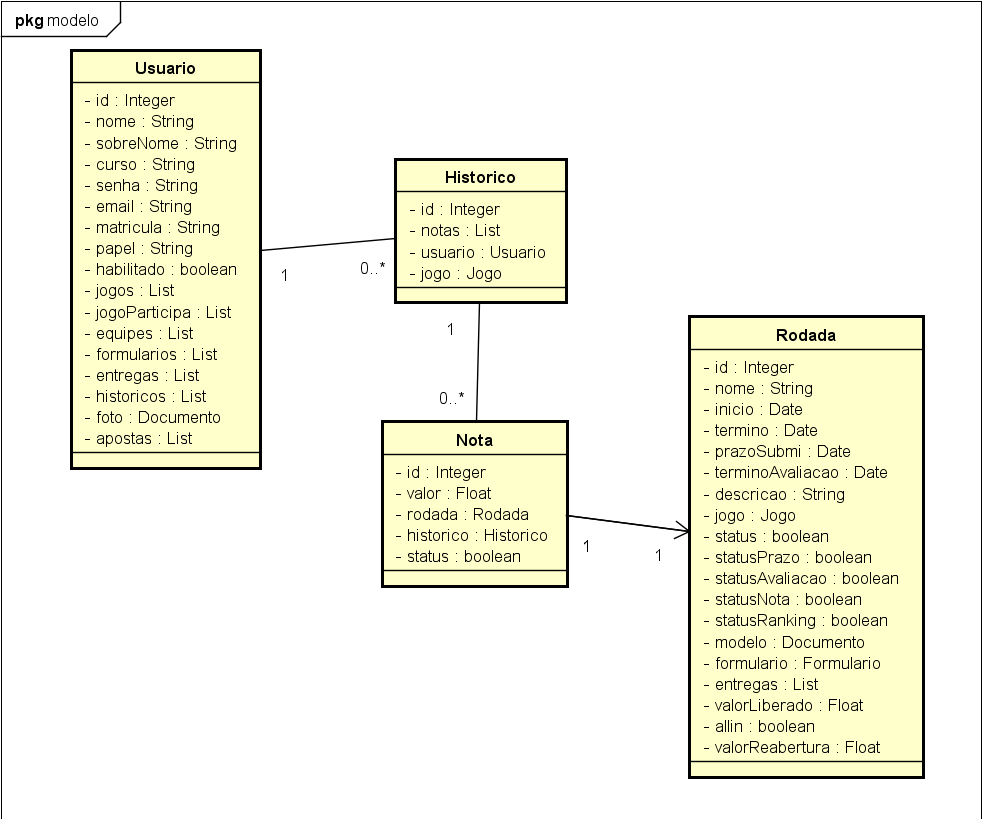
Figura 10 – Visão de apostas em uma rodada

****

Fonte: Produzido pelo autor

* 1. **Visão do histórico de notas de Usuario com Rodada**

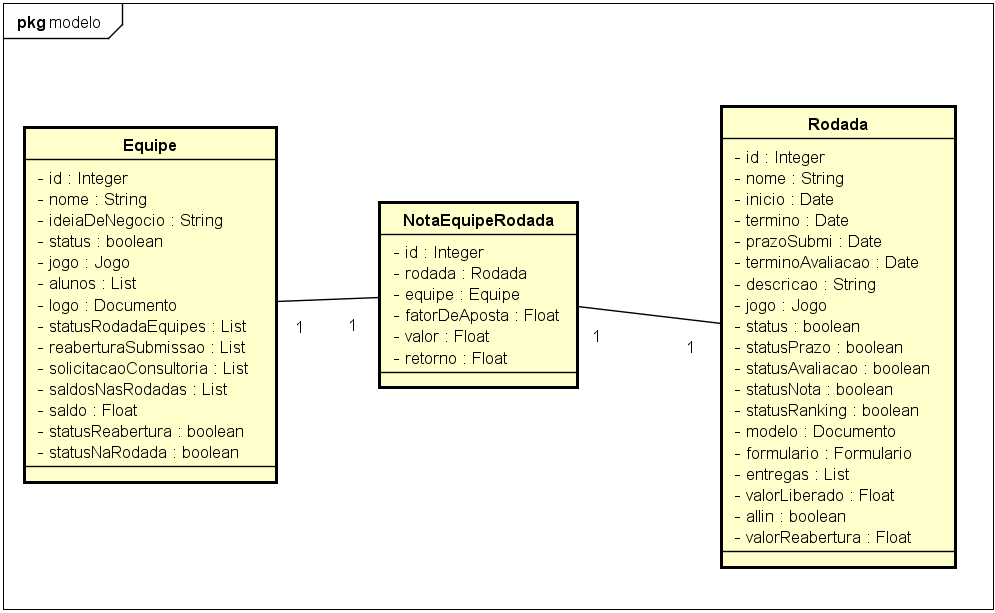
Figura 11 – Histórico - Usuário Rodada

****

Fonte: Produzido pelo autor

* 1. **Visão do relacionamento entre equipe e suas notas na rodada**

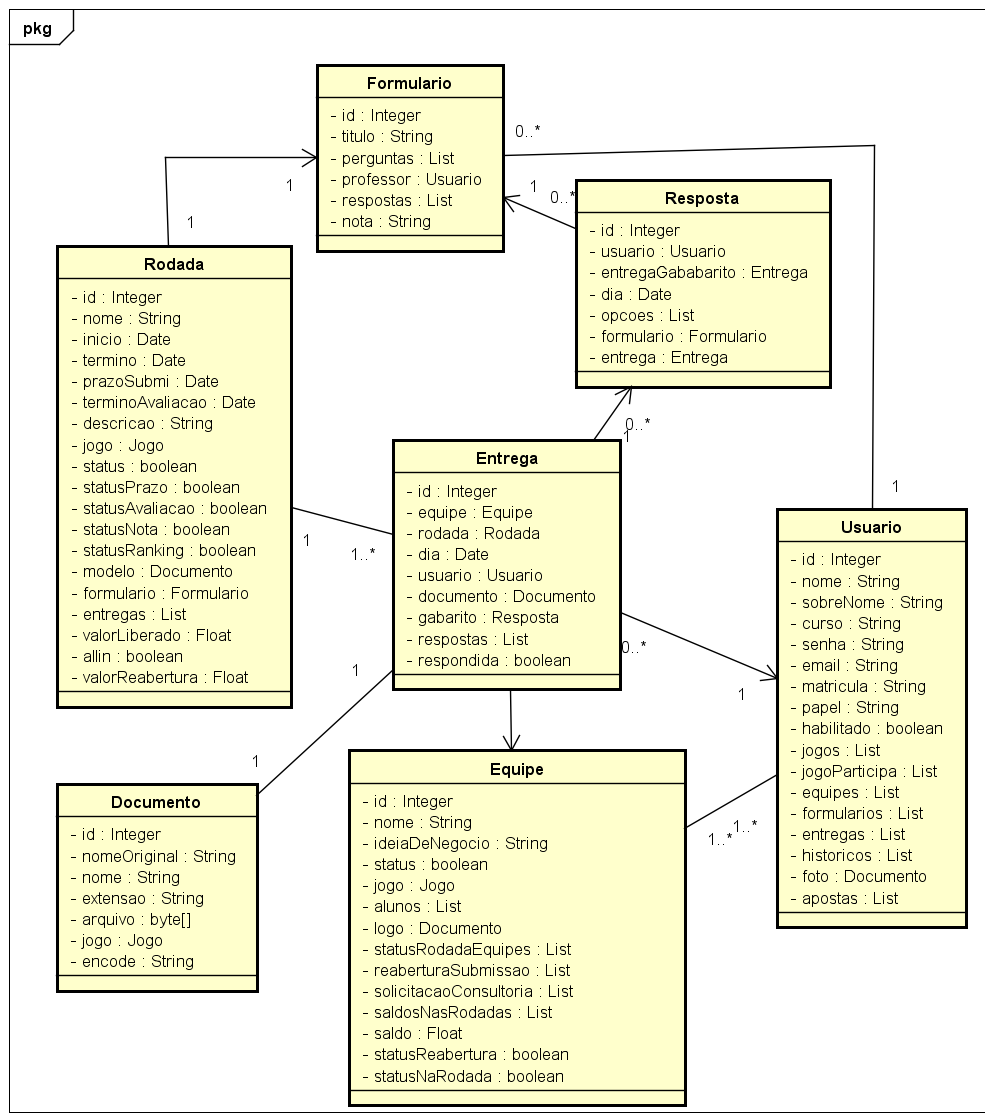
Figura 12 – Nota equipe rodada

****

Fonte: Produzido pelo autor

* 1. **Visão de solicitação de reabertura para rodadas**

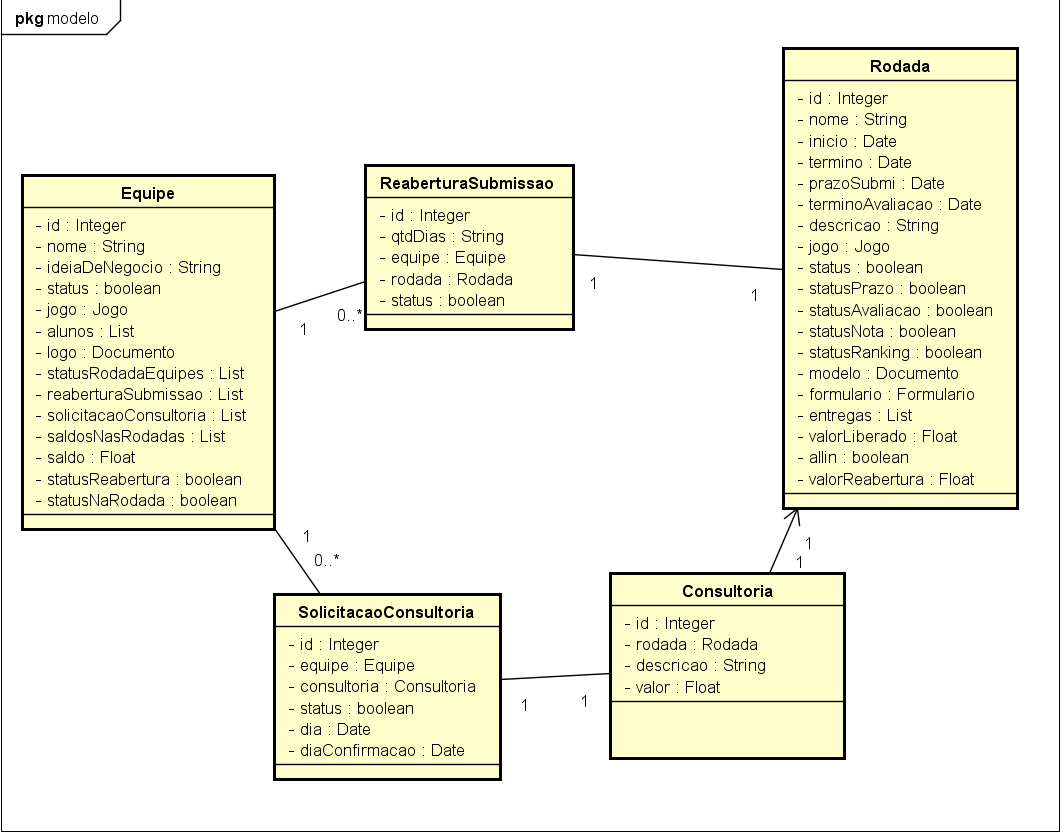
Figura 13 – Visão de entregas para uma rodada

****

Fonte: Produzido pelo autor

* 1. **Visão de serviços oferecidos pelo professor**

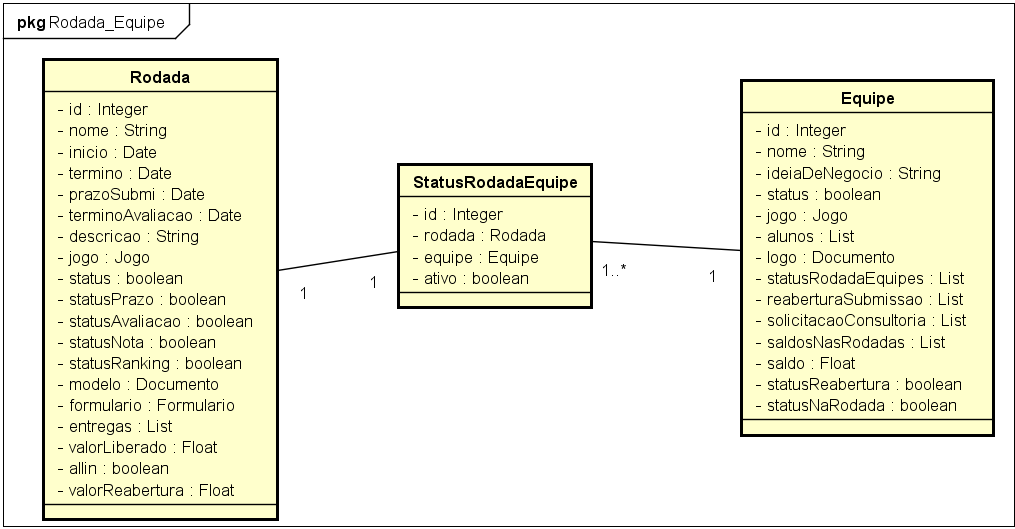
Figura 14 – Serviços de uma rodada

****

Fonte: Produzido pelo autor

* 1. **Status de equipe na rodada**

Figura 15 – Relacionamento status de uma equipe numa rodada

****

Fonte: Produzido pelo autor

1. **Qualidade**

* Modularidade:
  + Os componentes estão separados por seus estereótipos, cada classe pertencente ao sistema tem um pacote específico que agrupa classes que possuem o mesmo estereótipo.
* Modificabilidade:
  + Por possuir componentes com distribuição modular, separados por interesses, onde a camada de apresentação ou visão, negócio e gerenciamento de dados são encapsuladas, aumenta-se a modificabilidade. Onde pode-se alterar a lógica interna de algum componente sem que, em muitos casos, tenha-se impacto em outros níveis.
* Manutenibilidade:
  + Como há separação de interesses no momento de se criar um componente ou classe, seguindo padrões de nomenclatura, buscando manter a coesão entre pacotes, facilita-se assim a manutenibilidade da arquitetura.

**REFERÊNCIAS**

WEISSMANN, Henrique Lobo. **Vire o jogo com Spring Framework**. São Paulo, SP: Casa do Código, 2012. p. 133.