EDRA.com

Documento de Arquitetura

Grupo Pollux

2024.1

Versão 2.0

Histórico de Revisão

Data	Versão	Descrição	Autor(es)
13/05/2024	1.0	Considerações iniciais	Júlia Fortunato e Maria Clara
16/05/2024	1.0	Escrita dos tópicos 2.1, 2.2, 2.3 e o início do 2.5. Utilização das referências	Júlia Fortunato e Maria Clara
17/05/2024	1.0	Continuação da elaboração do documento	Maria Clara
18/05/2024	1.0	Finalização do item 2.5	Felipe Matheus
18/05/2024	1.0	Realização dos diagramas do tópico 2.6	Júlia Fortunato e Maria Clara
19/05/2024	1.0	Diagrama de classes e de pacote e tópico 2.8	Maurício e Kauã
19/05/2024	1.0	Finalização da escrita do documento	Maria Clara, Júlia, Felipe M., Kauã e Maurício
11/06/2024	2.0	Correção de estilo arquitetural	Júlia Fortunato
15/06/2024	2.0	Correção do Diagrama de casos de uso	Felipe Matheus

Autores:

Matrícula	Nome	Descrição do papel assumido na equipe	% de contribuição ao trabalho (*[1])
221008338	Maria Clara Oleari de Araújo	Product Owner	32,5
221022355	Júlia Rocha Fortunato	Scrum Master	32,5
221031274	Felipe Matheus Ribeiro Lopes	Product Owner	12%
222007021	Maurício Ferreira	Dev	11,5%
221022631	Kauã Richard	Dev	11,5%

Sumário:

1.0 INTRODUÇÃO

1.1 Propósito

Este documento tem como propósito, descrever a arquitetura do sistema sendo desenvolvido pelo grupo Pollux, na disciplina de MDS - Métodos de Desenvolvimento de Software (2024.1), EDRA.com, a fim de fornecer uma visão do sistema para desenvolvedores, testadores e demais interessados.

1.2 Escopo

O projeto EDRA.com visa criar um sistema para a equipe de competição da Universidade de Brasília (UnB) Campus Gama, a EDRA - Equipe de Robótica Aérea. Notabilizando sua necessidade de gerenciamento e divulgação. Após entrevistas com o cliente, foram identificadas as prioridades e funcionalidades essenciais. As principais tarefas incluem criar um *website* para divulgação e gerenciamento da equipe.

Demais detalhes com relação ao escopo do projeto, se encontram no documento "*Documento de Visão - Pollux*", na seção 4.

2.0 REPRESENTAÇÃO ARQUITETURAL

2.1 Definições

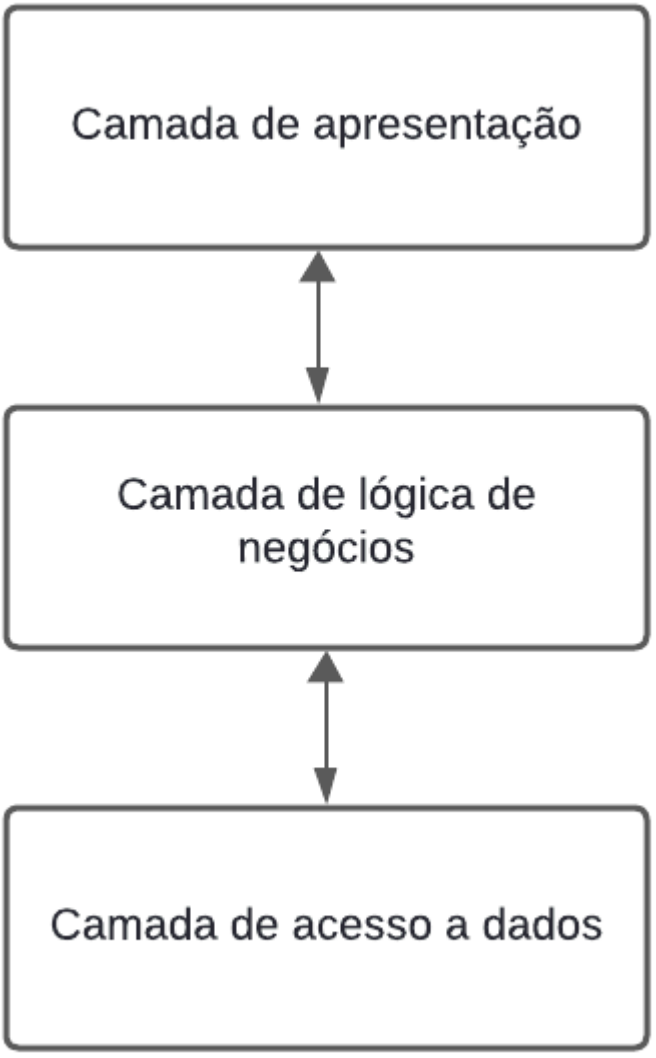
O sistema seguirá uma arquitetura em camadas, como apontado na Figura 1, em conjunto com o padrão de design de software MVC (Model View Controller) no seu desenvolvimento. É um padrão de arquitetura de software amplamente utilizado para desenvolver interfaces de usuário em aplicativos web. Funciona ao separar os componentes de um aplicativo em três partes principais. Conforme demonstrado também na Figura 1 *(MA et al., 2019)*

- **Camada de lógica de negócios (Model)**: Representa os dados e a lógica de negócios do aplicativo. O modelo é responsável por gerenciar o estado dos dados, realizar operações de leitura e escrita no banco de dados e executar a lógica de negócios.

- **Camada de apresentação (View)**: É responsável pela apresentação dos dados ao usuário. A visão é a interface com a qual o usuário interage e exibe as informações provenientes do modelo. Ela não possui lógica de negócios, apenas formata e exibe os dados.

- **Camada de acesso a dados (Controller)**: Atua como intermediário entre o modelo e a visão. Ele recebe as entradas do usuário, processa as solicitações, interage com o modelo para recuperar ou atualizar os dados necessários e decide qual visão deve ser apresentada ao usuário em resposta a essas entradas. Responsável pela interação com o armazenamento de dados, como bancos de dados ou serviços externos. Abstrai a complexidade do acesso aos dados para a camada de lógica de negócios, fornecendo uma interface consistente para recuperar e persistir informações.

Figura 1.Padrão arquitetural em camadas



2.2 Justifique sua escolha

No campo do desenvolvimento de software, a seleção de uma arquitetura apropriada pode definir o sucesso e a eficiência de um projeto. Entre os designs arquiteturais amplamente reconhecidos, o padrão Model-View-Controller (MVC) e a arquitetura em camadas destacam-se pela sua capacidade de separar claramente as diferentes responsabilidades dentro de um sistema. Conforme observado em *(MA et al., 2019):*

> *Um dos principais benefícios do padrão de design MVC é a separação da representação de informações e das estruturas de dados subjacentes. Isto suporta múltiplas visualizações para as mesmas informações, sem alterar as estruturas de dados subjacentes. Como a representação da informação (View) desconhece a lógica definida nos controladores, as visualizações poderiam ser reutilizadas em diferentes controladores.*

Considerando as vantagens proporcionadas por um modelo arquitetural no desenvolvimento do projeto a ser realizado, em aplicação web, mais detalhes no "*Documento de Visão - Pollux*". Portanto, a escolha estratégica da arquitetura em camadas com o design MVC (Model View Controller) em conjunto foi cuidadosamente deliberada, levando em consideração suas diversas vantagens, tais como:

- **Separação de Responsabilidades:** O MVC divide a aplicação em três componentes principais Isso permite uma clara separação de responsabilidades entre a lógica de negócios, a apresentação da interface do usuário e o controle das interações do usuário. Essa separação facilita a manutenção do código e o desenvolvimento de cada parte do aplicativo de forma independente.
- **Facilidade de Teste:** torna mais fácil testar cada componente do aplicativo de forma isolada, permitindo identificar e corrigir problemas de forma mais eficiente.
- **Manutenção Simplificada:** as atualizações e modificações podem ser feitas de forma fácil e segura, sem afetar outras partes.
- **Desenvolvimento Paralelo:** permite que equipes diferentes trabalhem em diferentes partes do aplicativo de forma independente, aumentando a eficiência do processo de desenvolvimento.
- **Escalabilidade:** novas funcionalidades podem ser adicionadas ou modificadas sem afetar o restante do sistema, tornando mais fácil adaptar.

2.3 Detalhamento

Adotar o padrão arquitetural em camadas com Model-View-Controller (MVC), no qual cada componente desempenha funções específicas com o objetivo de assegurar as

vantagens previamente mencionadas no item 2.2 deste mesmo documento.

Então, a camada de lógica de negócios (o componente Modelo) será responsável por representar os dados e a lógica. Ele incluirá classes e estruturas de dados que representam as entidades do sistema, bem como métodos para realizar operações de leitura, escrita e manipulação desses dados. O Modelo também será responsável por interagir com o banco de dados para recuperar e persistir informações.

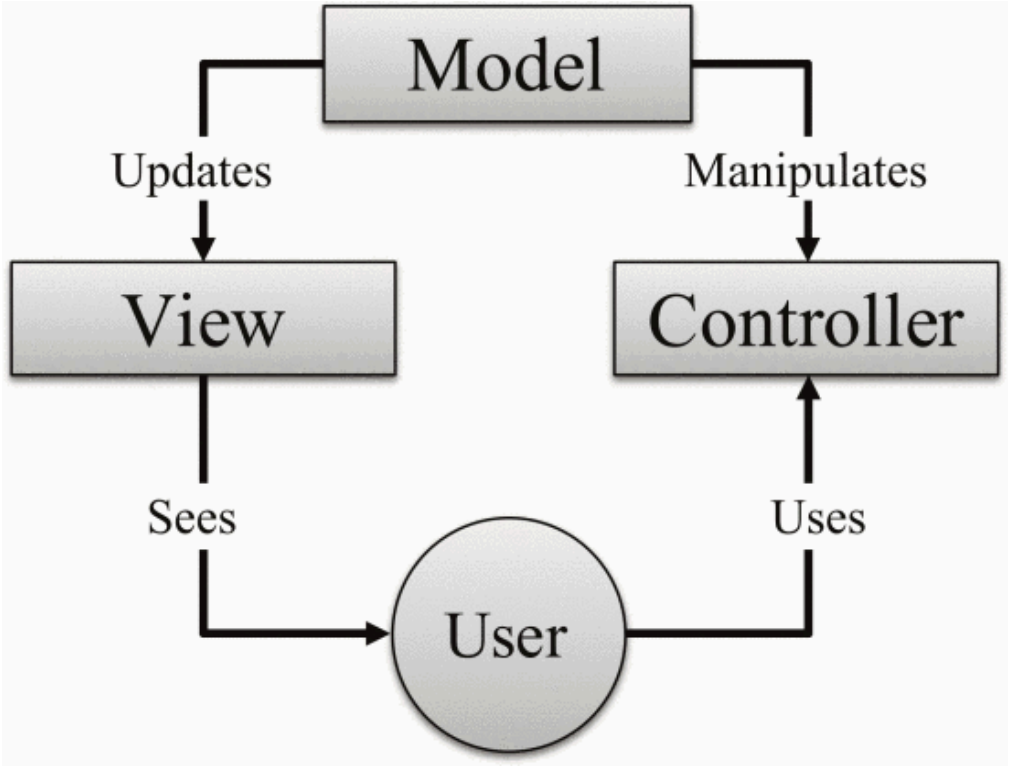
Já a camada de apresentação (o componente Visão) será responsável por apresentar os dados ao usuário de forma visualmente atraente e compreensível. Ele incluirá interfaces de usuário, páginas da web, elementos de design e estilos que permitirão aos usuários interagir com o sistema e visualizar as informações fornecidas pelo Modelo.

Por fim, a camada de acesso a dados (o componente Controlador) atuará como intermediário entre o Modelo e a Visão. Ele receberá as entradas do usuário, como cliques em botões ou preenchimento de formulários, e processará essas entradas para determinar as ações a serem tomadas. O Controlador então interage com o Modelo para executar as operações necessárias e decidirá qual Visão deve ser apresentada ao usuário em resposta às suas entradas.

A arquitetura em camadas combinada com o padrão MVC proporciona uma estrutura robusta, flexível e escalável para o desenvolvimento de aplicativos web. A clara separação de responsabilidades facilita a manutenção, o desenvolvimento paralelo e a escalabilidade do sistema, atendendo às necessidades do projeto de forma eficiente.

Portanto, a figura 2 abaixo mostra o esquema de funcionamento de um design MVC, como definido na seção 2.1 deste próprio documento. Nela, é possível visualizar a interação entre os componentes Modelo, Visão e Controlador, destacando a separação clara de responsabilidades e o fluxo de dados entre eles.

Figura 2.Padrão de design do controlador de visualização de modelo.



Fonte: *"Light-Weight and Scalable Hierarchical-MVC Architecture for Cloud Web Applications," 2019 6th IEEE*

2.4 Metas e restrições arquiteturais

Para definir as metas arquiteturais do software de divulgação e gerenciamento, considerar os seguintes aspectos:

- **Escalabilidade:** O sistema deve ser escalável para acomodar um crescente número de usuários e dados ao longo do tempo. A equipe pode gerar cada vez mais dados, exigindo uma infraestrutura que suporte, sem comprometer o desempenho.
- **Desempenho:** O software deve proporcionar um tempo de resposta rápido e um desempenho eficiente. Garantindo uma experiência satisfatória
- **Manutenibilidade:** A arquitetura do software deve facilitar a manutenção e

evolução do sistema ao longo do tempo. Permitindo assim, a implementação de novas funcionalidades, correções de bugs e melhorias contínuas de forma eficiente.

- **Segurança:** Proteger contra acessos não autorizados e garantir a integridade e confidencialidade dos dados.

Para definir as restrições arquiteturais do software de divulgação e gerenciamento, considerar o seguintes aspectos:

- **Compatibilidade**: O sistema deve ser compatível com todas as máquinas.

Essas metas e restrições arquiteturais devem ser cuidadosamente consideradas e incorporadas no processo de desenvolvimento para garantir que o software atenda às necessidades da equipe de competição e opere de maneira eficiente e segura.

2.5 Visão de Casos de uso (escopo do produto)

O projeto EDRA.com tem como objetivo desenvolver um sistema para a equipe EDRA, com foco em suas necessidades de gerenciamento e divulgação. A partir de entrevistas com o cliente, foram identificadas as prioridades e funcionalidades essenciais para o produto, por meio de um processo de brainstorming.

As principais tarefas prioritárias incluem a criação de um site de divulgação e gerenciamento da equipe. Para a parte de gerenciamento, o software deve ter controle financeiro, calendário com quadro *Kanban* e cronograma, controle de estoque, reuniões, redirecionamento para o Drive de documentos da equipe. Além disso, vai contar com diferenciação de logins para Membros e Capitães (cada um com suas permissões no sistema). Ademais, para a parte de divulgação, o site deverá ter apresentação da equipe, eventos e conquistas, informações sobre o processo seletivo e apresentação dos drones.

Os cenários funcionais definem os principais marcos do projeto, organizados em sprints, os quais ajudam a equipe de desenvolvimento a ter uma visão clara das funcionalidades a serem implementadas em cada etapa do projeto.

No diagrama abaixo, representado pela Figura 2, estão ilustrados os casos de uso da aplicação e o que cada um pode realizar de ação dentro do programa, de forma simplificada, a ser mais ilustrada ao decorrer do documento.

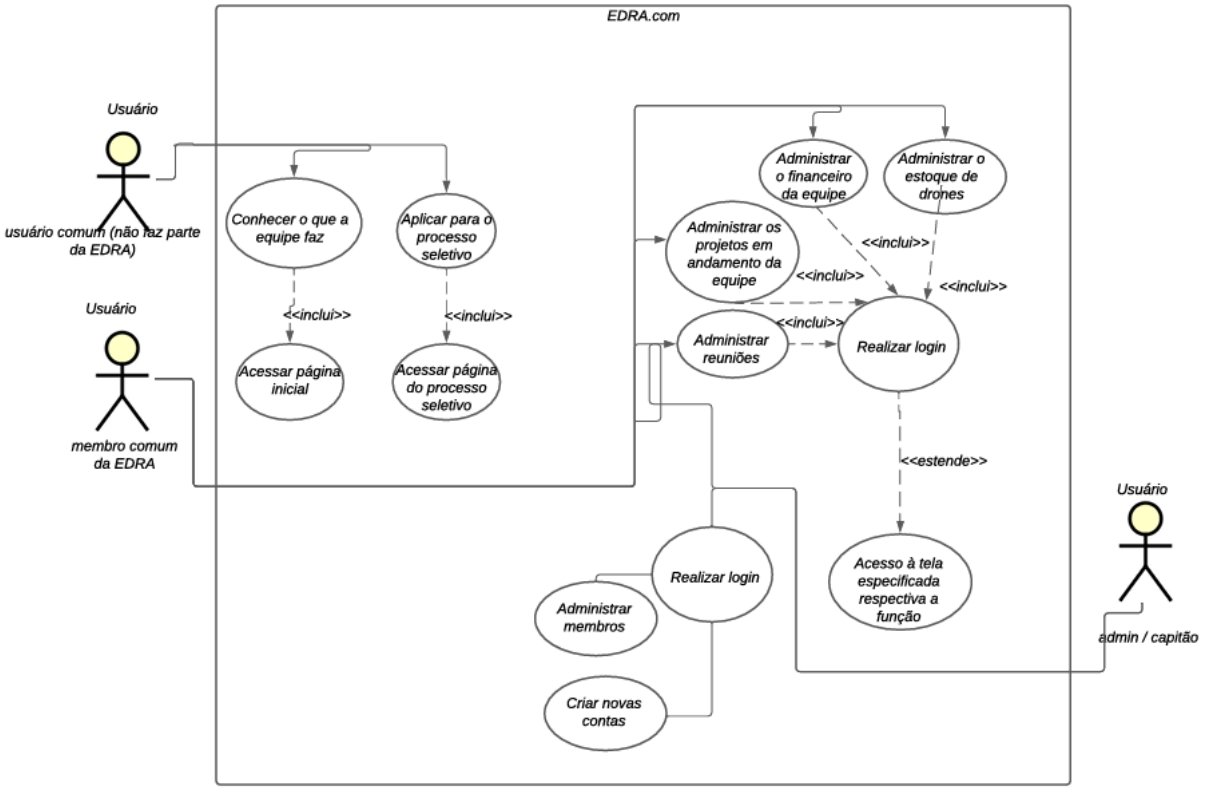


Figura 2 - Diagrama de casos de uso

As necessidades apresentadas pelo cliente são direcionadas ao público externo (usuários que não fazem parte da EDRA, mas demonstram interesse em conhecer ou participar da equipe) e ao público interno (usuários que fazem parte da EDRA). A diferença principal se baseia em que o público externo acessa a tela principal do site para acessar informações de divulgação , e se for do interesse do usuário, aplica para o processo seletivo na página específica, enquanto os membros da equipe acessam a sua função respectiva (administração de projetos, da equipe, do estoque de drones ou do financeiro) pela tela de login.

A interação entre o usuário e os dados utilizados auxiliaram na decisão de tomar o modelo MVC como a arquitetura do software, visto que a modularização entre as seções administrativas dos membros internos auxilia a manutenção. A utilização desse modelo também permite que os desenvolvedores trabalhem simultaneamente em áreas diferentes, facilitando a construção do software.

2.6 Visão lógica

O sistema é subdividido em dois grandes módulos:

- **Módulo de gerenciamento:** permite, somente membros da EDRA, criar conta, realizar o login e até mesmo editar os dados da conta. Ao fazer o login, aparecerá uma barra lateral de tarefas permitindo navegar entre as telas cronograma, reunião, estoque, financeiro e documentos.
  - Módulo de Calendário e Cronograma: permite a visualização e gestão de eventos, tarefas e prazos através de um quadro *Kanban* e de um calendário em formato de agenda;
  - Módulo de Reuniões: viabiliza adicionar novas reuniões, contendo a ata e a lista presença de cada uma das mesmas;
  - Módulo de Controle de Estoque: monitora a quantidade e o status (disponível e indisponível) das peças;
  - Módulo de Controle Financeiro: deverá selecionar o mês e o ano desejado, após essa ação aparecerá uma tabela mensal da respectiva data selecionada. Na mesma, irá conter espaço para descrição, entradas e saídas. O sistema calcula o valor final total e salva todas as respectivas informações. Possível adicionar novos dados na tabela;
  - Módulo de Redirecionamento para o Documentos: possibilita adicionar links para redirecionar a outros sites desejados que contenham documentos da equipe.

- **Módulo de divulgação:** proporciona a visualização, em uma única tela, para

todos os usuários interessados. Nesta tela, contém uma barra fixa para auxiliar a navegação durante todo o *scroll* da página, que contém todas as informações para conhecer a EDRA e fazer parte, caso tenha interesse, entre elas:

- Quem somos?: objetivo e apresentação da EDRA e das suas respectivas equipes internas, contendo também foto de todos os integrantes com seus respectivos nomes e equipe a qual participa;
- Nossos drones: fotos dos drones montados e pequena descrição a respeito dos mesmos;
- Processo seletivo: pequeno texto chamativo para entrar na equipe, disponibilização do cronograma do processo seletivo, do edital e do formulário para a inscrição do mesmo;
- Eventos e Competições: fotos com descrição sobre eventos e competições participadas, e uma tabela com as próximas, informando data e local.

Esses módulos se comunicam através de interfaces definidas, permitindo a troca de dados e informações entre eles de forma eficiente. Mais informações disponíveis no protótipo de alta fidelidade, presente no Figma e disponibilizado no repositório "*2024.1 - POLLUX*" na pasta *docs* e também será ilustrado no diagrama de estados da aplicação.

A seguir, na Figura 3, temos ilustrado o diagrama de estados da aplicação, que é aplicado para demonstrar o comportamento geral do sistema, mostrando os estados e transições do funcionamento do projeto - EDRA.com. Sendo assim, é possível entender o comportamento do sistema conforme ele é utilizado, com suas principais transições.

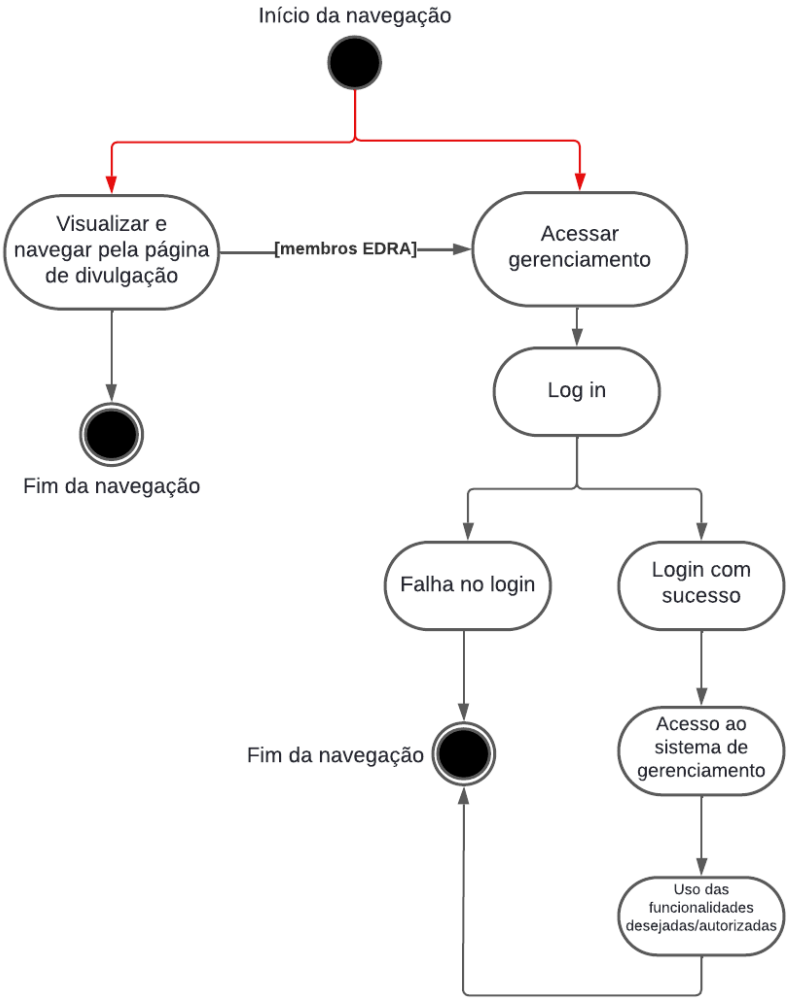


Figura 3 - Diagrama de Estados da aplicação

Além disso, para demais detalhes de como a aplicação funciona, inclui o diagrama de atividades do projeto, apresentado na Figura 4, que é aplicado para descrever as etapas realizadas conforme o usuário usa o sistema, deixando o funcionamento do mesmo mais claro. Sendo assim, no diagrama abaixo, é possível ver o que acontece em cada possibilidade conforme a utilização do site, desde quando o usuário abre o sistema e se depara com a tela de divulgação/login até quando o usuário já utilizou o gerenciamento da EDRA (caso membro).

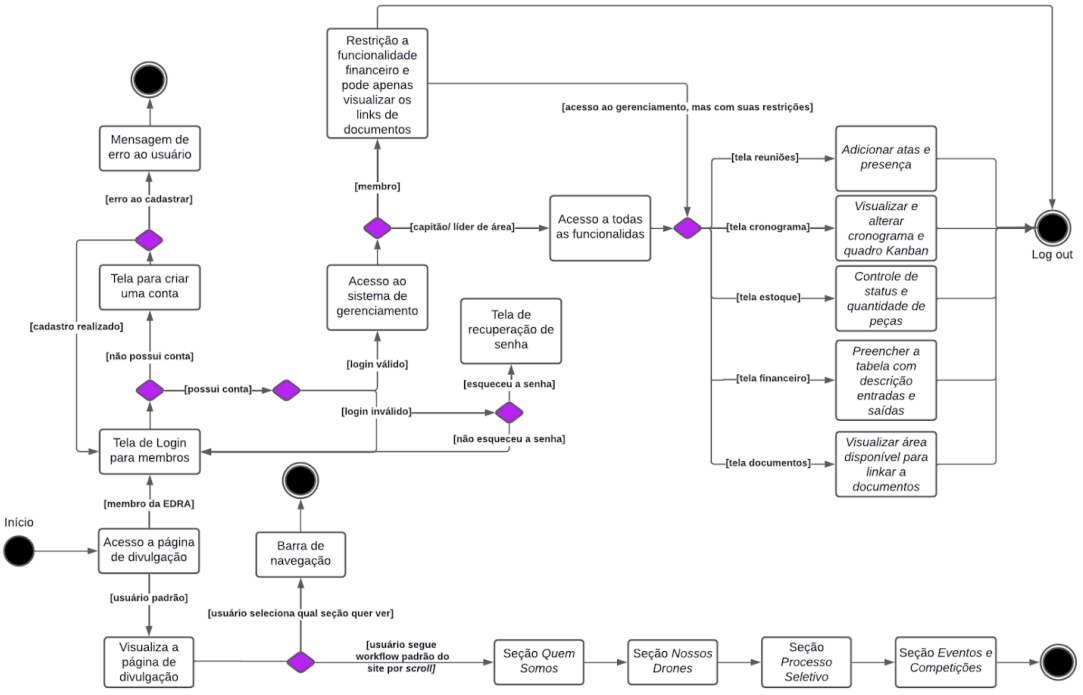


Figura 4 - Diagrama de Atividades da aplicação

Nele, também temos detalhadas as possíveis telas que participam da navegação do site, como proposto protótipo de alta fidelidade, disponível no GitHub e também no link a seguir: EDRA - protótipo. Além de, como é possível chegar em cada uma delas e como deve ser feita a navegação.

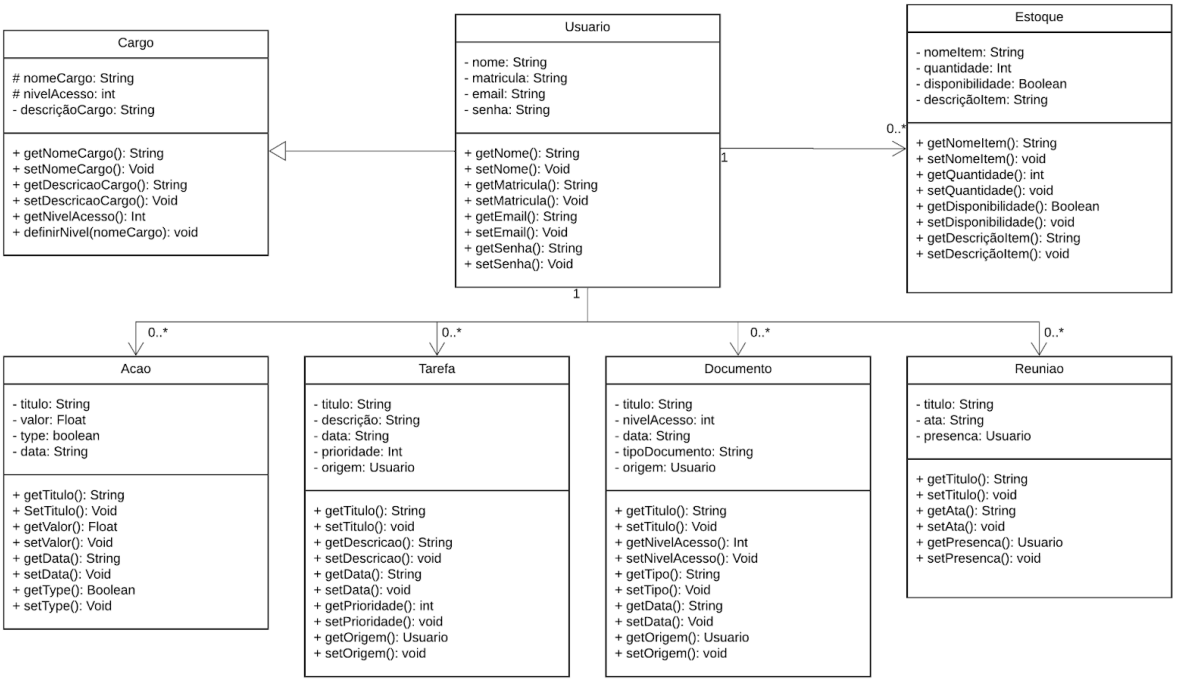


Figura 5 - Diagrama de Classes

O diagrama de classes para o gerenciamento da equipe de competição EDRA foi projetado para organizar e otimizar o controle de diversos aspectos envolvidos na administração da equipe. Este sistema compreende sete classes principais, cada uma com responsabilidades específicas que, em conjunto, contribuem para uma gestão eficiente e integrada.

2.7 Visão de Implementação

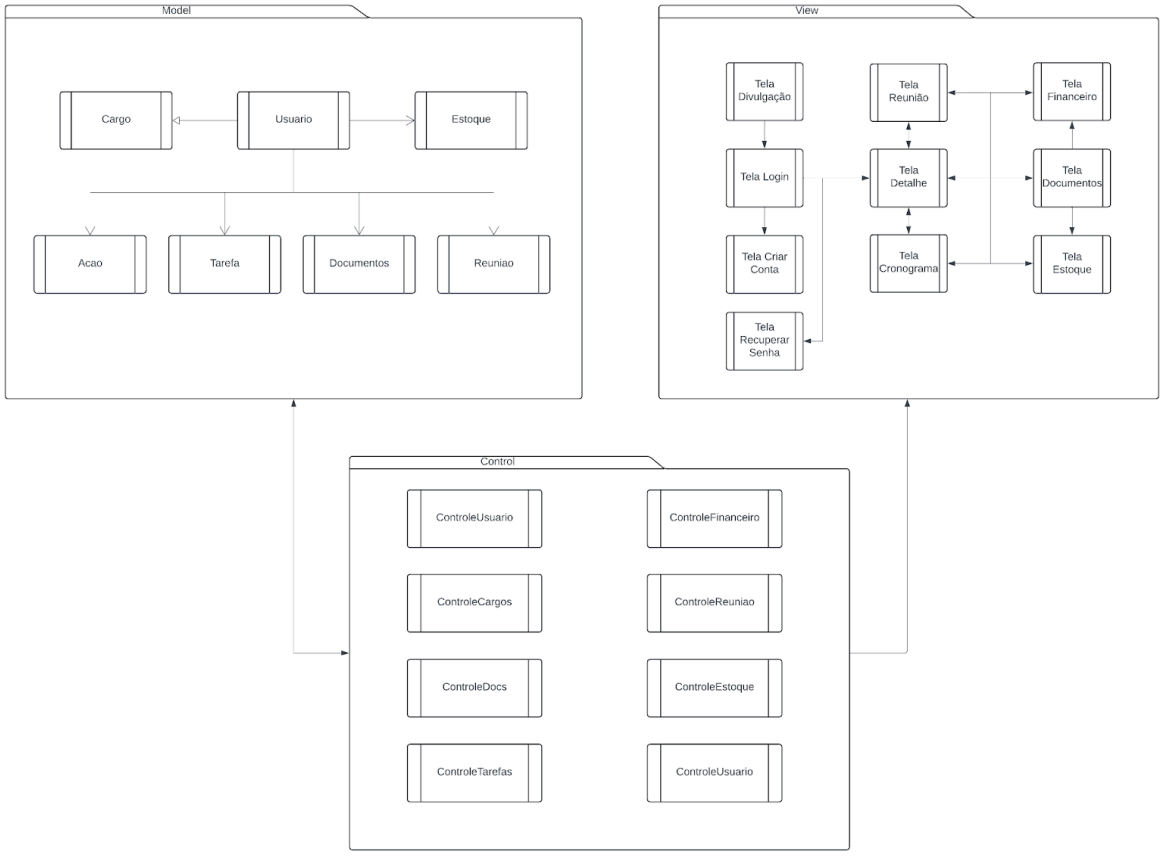


Figura 6 - Diagrama de pacotes

No diagrama de pacotes cada pacote representa uma área funcional distinta do sistema, o que simplifica o desenvolvimento, a manutenção e a compreensão do código-fonte. A organização promovida pelo diagrama de pacotes é crucial para garantir a coesão e o baixo acoplamento entre os componentes. Essa abordagem contribui significativamente para a robustez da aplicação, permitindo que novos recursos sejam adicionados com facilidade e minimizando o impacto de alterações em partes específicas do sistema.

2.7.1 Camada e apresentação

A camada de apresentação é responsável por fornecer interfaces que permitem a interação do usuário com os elementos do projeto. Essas interfaces possibilitam o acesso a todas as telas e ferramentas da aplicação. Em outras palavras, a camada de apresentação é o ponto de entrada para os usuários, permitindo que eles visualizem e interajam com os recursos disponíveis no sistema.

2.7.2 Lógica de negócios e regras de negócios

A lógica de negócios foi desenvolvida com base no paradigma orientado a objetos, oferecendo ao usuário a capacidade de gerenciar uma variedade de objetos inter-relacionados de acordo com as funcionalidades propostas. Através desse modelo, as regras de negócios são aplicadas de modo a restringir o acesso de determinados usuários a funcionalidades específicas. Essa abordagem permite uma organização eficiente e adaptável do sistema, garantindo que as operações sejam executadas de acordo com as políticas e procedimentos definidos.

2.7.3 Comunicação com banco de dados

Nossa comunicação com o banco de dados será facilitada pelas ferramentas Docker e Ruby on Rails. O Docker será responsável por manter o banco de dados funcionando de maneira eficiente e consistente, garantindo sua disponibilidade e confiabilidade. Por sua vez, o Ruby on Rails assumirá a responsabilidade pela gestão e manipulação dos dados, facilitando a comunicação entre o front-end e o banco de dados. Essa abordagem integrada proporciona uma infraestrutura sólida e uma camada de aplicação dinâmica, otimizando a operação do sistema e garantindo uma experiência fluida para o usuário.

2.8 Visão de Implantação

O software será implantado em um desktop para ser executado como um servidor web, seguindo o padrão das aplicações modernas. Para o front-end, utilizaremos as tecnologias React, Vite e JavaScript, que proporcionam rapidez e alta qualidade no desenvolvimento. A estruturação das páginas será feita com HTML e a estilização com CSS. No back-end, optamos pela linguagem Ruby, devido à sua excelente integração com a arquitetura MVC (Model-View-Controller) e a ferramenta Rails. Essa escolha garante uma ótima compatibilidade com Ruby, MySQL e o servidor Apache, promovendo agilidade e eficiência no projeto, além de minimizar possíveis problemas de incompatibilidade entre as ferramentas.

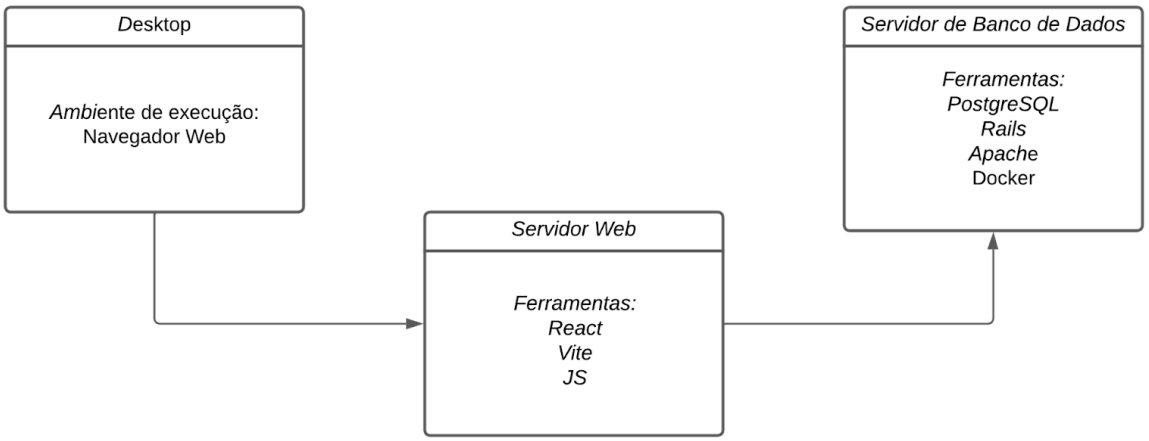


Figura 7 - Diagrama da visão de implantação do produto

2.9 Restrições adicionais

O software possui restrições específicas relacionadas a aspectos negociais e de qualidade que devem ser observadas rigorosamente para garantir o seu correto funcionamento e a segurança dos dados.

- O site deve ser acessível a todos os interessados, permitindo uma ampla divulgação das atividades e conquistas da equipe.
- O software de gerenciamento exige autenticação e login dos membros da equipe EDRA. Essa medida é essencial para controlar o acesso ao sistema e manter a integridade das operações e dados da equipe.
- Alguns setores do software terão o acesso limitado a determinados membros.
  - A aba do financeiro da equipe só poderá ser acessada por membros que compõem essa seção.
  - A aba dos documentos só poderá ser visualizada pelos membros.
  - Os capitães têm acesso irrestrito a todas as partes do software e poderão adicionar conteúdo na aba de documentos.

Para garantir um desempenho excelente e atender às expectativas dos usuários, as seguintes características de qualidade são cruciais e devem ser mantidas durante todo o ciclo de vida do projeto:

- **Usabilidade:** A interface deve ser intuitiva e fácil de usar, proporcionando uma experiência agradável aos usuários e facilitando a navegação e o uso das funcionalidades do sistema.
- **Confiabilidade:** O sistema deve apresentar alta disponibilidade, garantindo que os usuários possam acessá-lo sempre que necessário, além de contar com mecanismos eficazes de recuperação rápida em caso de falhas.
- **Portabilidade:** O software deve ser compatível com diferentes dispositivos e sistemas operacionais, permitindo que os usuários acessem o sistema de maneira conveniente, independentemente do equipamento ou plataforma que estejam utilizando.
- **Segurança:** Deve haver proteção contra acessos não autorizados, assegurando a integridade dos dados e a privacidade das informações dos usuários. Implementar medidas de segurança robustas é essencial para proteger o sistema contra ameaças e ataques.

3.0 BIBLIOGRAFIA

M. Ma, J. Yang, P. Wang, W. Liu and J. Zhang, "Light-Weight and Scalable Hierarchical-MVC Architecture for Cloud Web Applications," 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), Paris, France, 2019, pp. 40-45, doi: 10.1109/CSCloud/EdgeCom.2019.00017. keywords: {Service-oriented architecture;Scalability;Cloud computing;Computer architecture;Web pages;HTML;Cascading style sheets;Web Application;Cloud Computing;MVC;Modularization}