* **Entrega 1 (Peso 5,0) — Documentação parcial da arquitetura** conforme os modelos e exemplos anexados à atividade do Google Classroom. Este documento deve conter ao menos:
  + Descrição do escopo do sistema (Luca Machado)
  + Descrição dos requisitos arquiteturais, objetivos e restrições da arquitetura (João Pedro)
  + Definição dos padrões arquiteturais adotados (Luca Machado)
  + Justificativa das decisões arquiteturais adotadas com base nos atributos de qualidade do sistema. Descrição de como a arquitetura de software contribui para os atributos de qualidade importantes do sistema (Luca Machado)
  + Documentação das visões arquiteturais
    - O modelo conceitual (Daniel e Luca)
    - Ao menos diagramas de casos de usos (Luca e Pedro Paradela), modelo conceitual e diagramas de sequência do sistema (Se o sistema for grande, não precisa fazer os DSS dos CRUDs. Mínimo 1 DSS (diagramas de sequência do sistema) por integrante da equipe)
      * Cadastro de conta (Luca Machado)
      * Atualizar Dados Cadastrais (Luca Machado)
      * Aprovar certificado (Pedro Paradela)
      * Contratar Serviço(Rodrigo)
      * Banir usuário (Luca)
      * Aceitar contratação (Luca)
      * Avaliar Serviço (Daniel)
      * Envio de Mensagens(Rodrigo)
      * Acessar histórico (Rodrigo)

-Diagrama com a visão geral do sistema (organização), incluindo os estilos arquiteturais utilizados e justificativas de uso - (Pedro Paradela)

* + Diagramas de interação. Pelo menos um diagrama por caso de uso. Escolher a operação do DSS mais complexa. Obs: Descrever o contrato da operação escolhida. (Mínimo 1 diagrama por integrante da equipe)
    - Cadastro de conta (Luca Machado)
    - Atualizar Dados Cadastrais (Luca Machado)
    - CRUD anúncios (Rodrigo)
    - CRUD certificado (Rodrigo)
    - Aprovar certificado (Pedro Paradela)
    - Contratar Serviço(Rodrigo)
    - Banir usuário (Luca)
    - Aceitar contratação (Luca)
    - Avaliar Serviço (Daniel)
    - Envio de Mensagens(Rodrigo)
    - Acessar histórico (Rodrigo)
  + Diagramas de classe detalhado(Luca)
  + slides (João Pedro)

REVISÕES PÓS PRIMEIRA ARQUITETURA

- Casos de uso (Luca)

- Diagrama de classes geral (Luca)

- Slides (Luca)

- Arquitetura geral do sistema (Pedro Paradela)

- diagramas (Pedro Paradela 1, Luca Machado 5, Rodrigo 4)

**Servito**

**Documento de Arquitetura de Software**

**Versão <1.0>**

**Histórico da Revisão**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versão** | **Descrição** | **Autor** |
| 18/05/2025 | 1.0 | Primeira versão contendo os detalhes gerais da arquitetura junto com casos de uso e seus respectivos diagramas de Iteração | Luca Machado, Pedro Paradela, Rodrigo Chimelli, Daniel Molina, João Pedro |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## 

## Índice Analítico

1. Introdução 4

1.1 Finalidade 4

2. Metas e Restrições da Arquitetura 4

2.1 Metas 5

2.2 Restrições 5

3. Requisitos Arquiteturalmente Significantes 6

4. Mecanismos Arquiteturais 6

5. Visões da Arquitetura 7

5.1 Diagrama Geral 8

5.2 Casos de Uso 9

5.3 Modelo Conceitual 9

5.4 Diagramas de Sequência 10

5.5 Diagramas de Comunicação 11

5.6 Diagrama de Classe

**Documento de Arquitetura de Software**

## Introdução

Este é o documento de arquitetura de software do projeto “servito”, que consiste em uma aplicação web que permite que os usuários pessoas físicas, após se cadastrarem, anunciem e contratem serviços dos mais variados tipos (Reparos domésticos, design gráfico, aulas particulares, etc…). Os usuários poderão fazer buscas dos anúncios pelo tipo, localidade e nome do serviço. O site contará com um sistema de avaliação para os contratantes avaliarem os serviços dos prestadores. Também será possível que os usuários enviem certificados/diplomas para a equipe do site poder avaliá-los e confirmar a veracidade, passando mais credibilidade e confiança ao fazer o anúncio de seus serviços. O site também contará com moderação, que irá banir usuários que desrespeitarem as regras do site.

### Finalidade.

A finalidade deste documento é fornecer uma representação detalhada e estruturada da arquitetura do sistema “Servito”, fazendo com que as decisões arquiteturais (sobretudo, padrões de projeto) relevantes sejam aqui registradas, bem como todos os diagramas de apoio (Iteração, diagrama de classes, casos de uso etc). Esses registros têm como objetivo atuar como guia para tanto a atual equipe começar desenvolvimento e manter qualidade técnica satisfatória que siga os padrões GRASP como demais boas práticas de programação no geral. Além disso, este documento irá servir como referência para atuais e futuros colaboradores poderem expandir as funcionalidades do sistema e para manutenção geral do mesmo.

## Metas e Restrições da Arquitetura

A ideia da arquitetura do "Servito" é a de construir uma plataforma de marketplace de serviços que seja confiável, escalável e intuitiva, priorizando a segurança dos dados e a facilidade de uso para prestadores e contratantes de serviços.

Questões que orientam a arquitetura:

* O sistema deverá lidar com um grande número de anúncios e interações entre usuários e sua integração com o sistema.
* É necessário garantir a integridade dos dados utilizados no sistema de avaliações e na verificação de credenciais para que os usuários confiem na plataforma e a utilizem sem restrições.
* O sistema precisa ser bem estruturado e com estruturas bem definidas e pensadas, para suportar atualizações contínuas, com a possível adição de serviços, funcionalidades e um aumento de usuários ao longo do tempo, sem comprometer a manutenção do sistema.
* O sistema precisa oferecer um algoritmo de busca relevante para o usuário e eficiente, mesmo com um volume alto de anúncios e usuários.

Questões críticas que a arquitetura deve atender:

* **Dependência de Armazenamento de Arquivos (Certificados):** A arquitetura deve prever um mecanismo seguro e eficiente para o upload, armazenamento e acesso aos arquivos de certificados/diplomas, isolando essa funcionalidade para não sobrecarregar outros componentes do sistema.
* **Performance da Busca:** Com o crescimento da base de anúncios, garantir que a funcionalidade de busca por tipo, localidade e nome do serviço permaneça rápida e relevante.
* **Gerenciamento de Estado e Sessão de Usuários:** A arquitetura deve gerenciar de forma eficiente as sessões dos usuários em uma aplicação web potencialmente distribuída.
* **Comunicação entre Componentes/Módulos:** Definir interfaces claras e eficientes para a comunicação entre os diferentes módulos do sistema (ex: como o módulo de anúncios interage com o módulo de usuários e avaliações)..

### Metas

* **Modularidade e Baixo Acoplamento:** Isolar as principais funcionalidades (gestão de usuários, anúncios, busca, avaliações, verificação de credenciais, moderação, sistema de mensagens) em componentes ou módulos bem definidos, permitindo desenvolvimento, manutenção e escalabilidade independentes.
* **Escalabilidade:** A arquitetura deve ser capaz de suportar um aumento significativo no número de usuários, anúncios e transações sem degradação de performance. Isso inclui a capacidade de escalar horizontalmente os componentes do backend.
* **Segurança:** Proteger os dados pessoais dos usuários, as informações de anúncios, as mensagens trocadas e, crucialmente, o processo de verificação de credenciais. A arquitetura deve prever mecanismos robustos de autenticação, autorização e proteção contra vulnerabilidades comuns.
* **Usabilidade e Desempenho:** Oferecer uma interface de usuário responsiva e intuitiva, com tempos de resposta rápidos para as principais operações, como busca de serviços, visualização de anúncios e processo de contratação.
* **Manutenibilidade e Extensibilidade:** A estrutura do código e da arquitetura deve facilitar a correção de bugs, atualizações e a adição de novas funcionalidades no futuro com o mínimo de impacto em componentes existentes.
* **Confiabilidade:** Garantir que o sistema esteja disponível e funcione corretamente, especialmente as funcionalidades críticas como cadastro, busca, contratação e o sistema de avaliações.
* **Integridade dos Dados:** Assegurar a consistência e a validade dos dados armazenados, principalmente em relação aos perfis de usuário, anúncios, avaliações e o status de verificação de credenciais.

### Restrições

* **R01:** O sistema deve ser acessível em navegadores web, desktops e dispositivos móveis.
* **R02:** Todas as informações pessoais devem ser protegidas. A lógica de segurança será implementada no acesso ao Modelo e nos Controladores.
* **R03:** Limitações de tempo para desenvolvimento inicial podem influenciar a granularidade dos componentes e a escolha das tecnologias.
* **R04:** A estrutura de classes deve ser flexível para acomodar possíveis futuras expansões.

## Requisitos Arquiteturalmente Significantes

Os seguintes requisitos são considerados arquiteturalmente significantes para o "Servito", pois impõem demandas cruciais sobre a estrutura, o comportamento e as qualidades do sistema, influenciando diretamente as decisões de design arquitetural:

* **RF-Cadastro multifacetado de Usuário:** A capacidade de usuários se cadastrarem, gerenciamento de informações pessoais, exige uma modelagem de dados flexível e mecanismos de autenticação e autorização robustos. Impacta: Modelagem de Dados (Model), Segurança, Gerenciamento de Identidade.
* **RF-Gerenciamento Completo de Anúncios:** Permitir que prestadores criem, editem, e gerenciem seus anúncios de serviços, incluindo detalhes, tipos de serviço e possivelmente imagens ou portfólios. Impacta: Modelagem de Dados, Lógica de Negócios, Armazenamento de Dados e Arquivos.
* **RF-Busca Avançada e Performática:** A funcionalidade de busca por tipo, localidade e nome do serviço deve ser eficiente e retornar resultados relevantes rapidamente, mesmo com um grande volume de anúncios. Impacta: Otimização de Consultas, Potencial necessidade de um mecanismo de busca dedicado
* **RF-Sistema de avaliação:** Contratantes devem poder avaliar serviços, e essas avaliações devem construir um sistema de análises visível e confiável para os contratantes. Impacta: Modelagem de Dados, Integridade dos Dados.
* **RF-Verificação de certificados:** O processo de envio, análise e validação de certificados/diplomas pela equipe do site. Impacta: Fluxo de Trabalho de Moderação, Armazenamento Seguro de Arquivos, Interface para a equipe de moderação, Atualização de status do prestador.
* **RNF-Segurança de Dados**: A proteção de dados pessoais, informações de anúncios, mensagens e credenciais é primordial. Impacta: Todas as camadas da arquitetura, especialmente autenticação, autorização, armazenamento de dados e comunicação.
* **RNF-Escalabilidade:** O sistema deve ser capaz de lidar com um número crescente de usuários, anúncios e interações sem degradação de desempenho. Impacta: Design dos componentes Model e Controller, Infraestrutura de hospedagem, Estratégias de banco de dados.
* **RNF-Usabilidade e Desempenho da Interface:** A plataforma deve ser intuitiva e responsiva para todos os tipos de usuários em diferentes dispositivos. Impacta: Design da View, Otimização do frontend, Eficiência das APIs.
* **RNF-Disponibilidade e Confiabilidade:** O sistema deve estar acessível e operacional de forma consistente. Impacta: Infraestrutura de hospedagem, Design para tolerância a falhas.

## Mecanismos Arquiteturais

**MVC (Model-View-Controller)**

O padrão MVC (Model-View-Controller) é uma forma de organizar o código separando a aplicação em três partes principais: o Model, que representa os dados e a lógica de negócio, a View, que é responsável por exibir as informações ao usuário, e o Controller, que atua como intermediário, interpretando as ações do usuário, atualizando o modelo e escolhendo a view apropriada. Uma das principais vantagens desse padrão arquitetural é que ele separa as principais responsabilidades de 1 componente em 3, reduzindo drasticamente o acoplamento

O MVC se alinha fortemente com as necessidades arquiteturais e de qualidade do projeto, citando alguns pontos:

**Manutenibilidade e testabilidade:** Com a separação dos componentes nas 3 partes, tanto a manutenção quanto a condução de teste é facilitada, pois pode ser direcionada a um dos componentes relevantes (por exemplo, se for alterar a lógica de negócios, as camadas de view e o controller provavelmente não serão afetadas). Da mesma forma, testes unitários conseguem ser direcionados a um das 3 partes

**Escalabilidade:** Com a modularização que o MVC proporciona uma facilidade na expansão de funcionalidades de módulos já existentes, assim como a criação de novos. Isso ocorre de forma incremental e sem a necessidade de constantemente redesenhar o sistema ou modificar componentes já estabelecidos.

**Baixo acoplamento:** Como já dito anteriormente, a separação do que seria um único componente em model, view e controller, dividindo as responsabilidades que girariam em torno de um componente em 3, evitando que um só componente tenha muitas responsabilidades e consiga focar em uma parte específica do que tange a sua existência

## Visões da Arquitetura

### Diagrama Geral do Sistema

**A diagram of a company

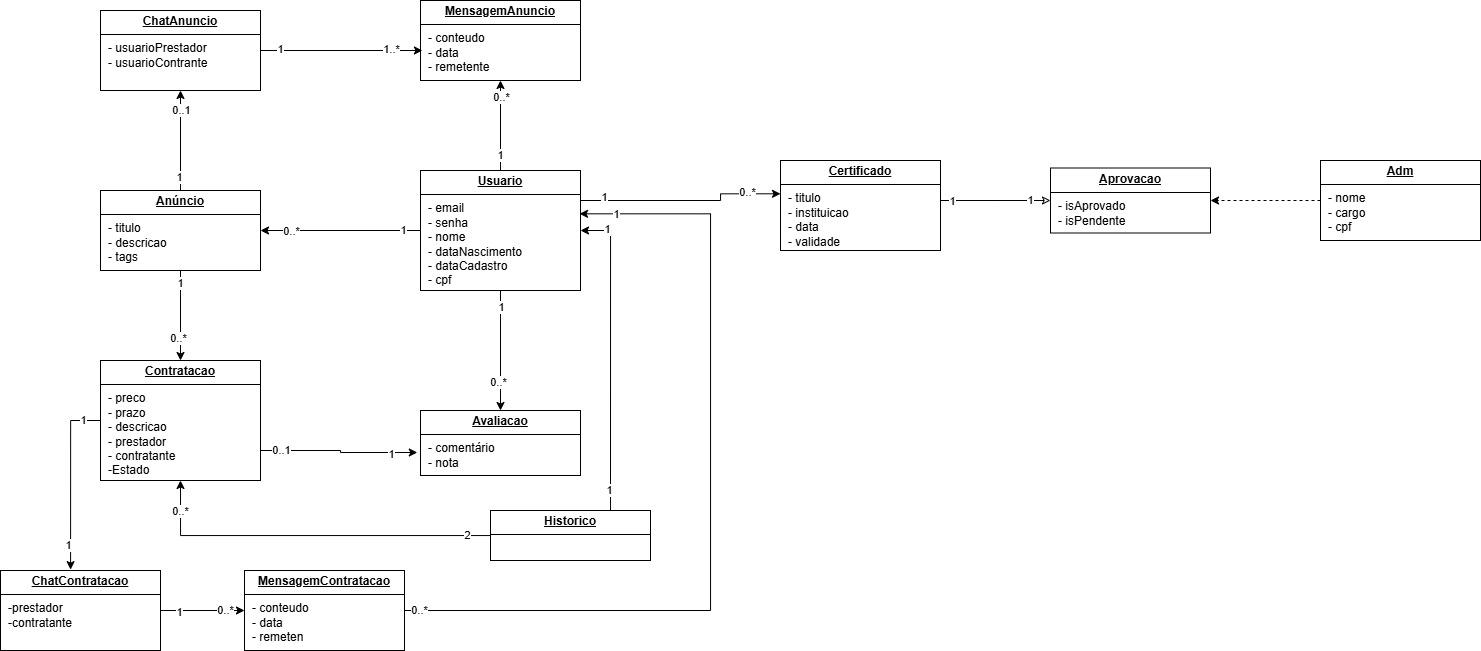
AI-generated content may be incorrect.**

### Casos de uso

**A diagram of a diagram

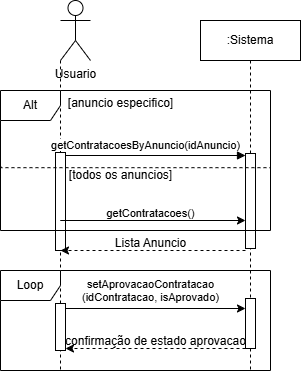
AI-generated content may be incorrect.**

### Modelo conceitual

****

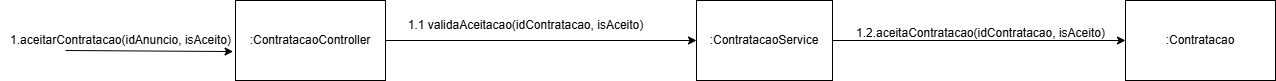
### Diagramas de Sequência de sistema

Foram feitos 1 diagrama de sequência de sistema para cada caso de uso (exceto os CRUDs). Abaixo, segue o exemplo para o caso de uso “Aceitar contratação”. Os demais diagramas se encontram na pasta de documentação do projeto no github



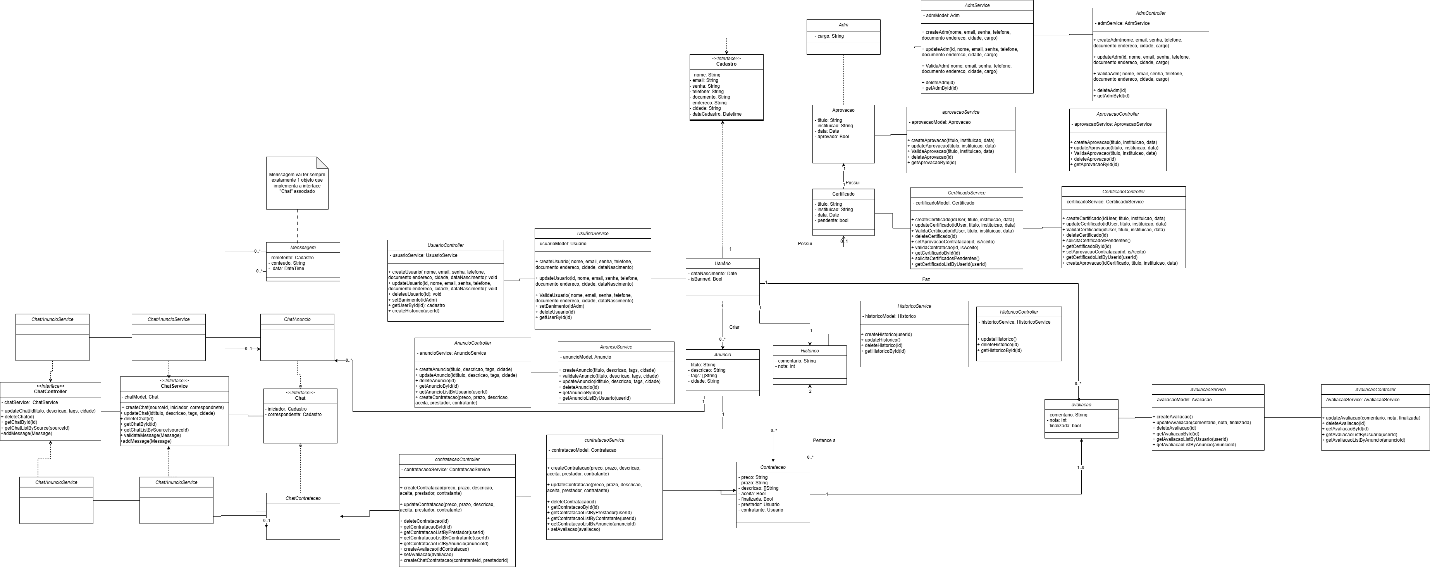
### Diagramas de comunicação

Foram feitos 1 diagrama de comunicação para cada caso de uso. Abaixo, segue o exemplo para o caso de uso “Aceitar contratação”. Os demais diagramas se encontram na pasta de documentação do projeto no github



### Diagrama de classe

Para melhor visualização, o diagrama de classes se encontra na pasta de documentação no projeto no github

**