# DOCUMENTO DE ESPECIFICAÇÃO E MODELAGEM DO SISTEMA SHARECONDO

**Projeto:** ShareCondo

**Integrantes:** Caio Rocha, Guilherme Spinola

**Disciplina:** Linguagem de Programação II & Modelagem de Software

**Semestre:** 6º

**Data:** 01 de junho de 2025

## Sumário

1. Introdução

1.1. Propósito do Documento

1.2. Escopo do Projeto

1.3. Definições, Acrônimos e Abreviações

1.4. Referências

1.5. Visão Geral do Documento

1. Descrição Geral

2.1. Perspectiva do Produto

2.2. Funções do Produto

2.3. Características dos Usuários

2.4. Restrições

2.5. Suposições e Dependências

1. Requisitos Específicos

3.1. Requisitos Funcionais

3.2. Requisitos Não-Funcionais

3.3. Requisitos de Interface

3.4. Requisitos de Dados

1. Modelagem de Processos de Negócio (BPMN)
2. Modelagem UML e Diagramas

5.1. Modelo de Domínio

5.2. Diagrama de Casos de Uso

5.3. Diagrama de Sequência

5.4. Diagrama de Classes Detalhado

1. Protótipos e Wireframes

6.1. Wireframes

6.2. Storyboard

6.3. Story Map

1. Análise de Prioridade de Recursos (Modelo Kano)
2. Casos de Teste Funcionais
3. Matrizes de Rastreabilidade

9.1. Matriz de Rastreabilidade: Requisitos vs. Objetivos

9.2. Matriz de Rastreabilidade: Requisitos vs. Casos de Teste

1. Glossário
2. Referências Bibliográficas e Ferramentas

## 1. INTRODUÇÃO

### 1.1 Propósito do Documento

Este documento tem como objetivo definir e especificar os requisitos funcionais, não-funcionais, de interface e de dados do sistema **ShareCondo**. Ele servirá como base para o desenvolvimento, testes e validação do produto, estabelecendo um acordo entre as partes interessadas sobre o que o sistema deverá fazer. [cite: 2, 12]

### 1.2 Escopo do Projeto

O ShareCondo é uma plataforma projetada para conectar vizinhos dentro de um condomínio, permitindo que eles ofereçam ou solicitem diversas habilidades, serviços e itens. O objetivo principal é fomentar um ambiente colaborativo, onde os moradores possam realizar trocas, empréstimos ou contratar pequenos serviços entre si, como reparos, jardinagem, aulas particulares, ou prática de idiomas.

**Dentro do Escopo:**

Gerenciamento de Usuários (Cadastro, Login, Aprovação de Moradores).

Gerenciamento de Condomínios (Criação e listagem).

Gerenciamento de Anúncios (Criação, listagem, visualização dos próprios anúncios).

Gerenciamento de Ofertas (Criação de ofertas para anúncios, aceitação/recusa de ofertas).

Sistema de autenticação e autorização baseado em papéis (ADMIN, SINDICO, USUARIO).

Interface front-end para interação do usuário e administração.

API back-end para processamento das lógicas de negócio e persistência de dados.

**Fora do Escopo:**

Sistema de pagamento integrado para transações monetárias (as negociações financeiras são de responsabilidade dos usuários).

Sistema de avaliação e reputação de usuários.

Comunicação em tempo real (chat) entre usuários (interações são baseadas em ofertas e dados de contato).

Aplicativo móvel nativo (o foco é uma aplicação web responsiva).

### 1.3 Definições, Acrônimos e Abreviações

| Termo | Definição |
| --- | --- |
| RF | Requisito Funcional [cite: 16] |
| RNF | Requisito Não-Funcional [cite: 16] |
| RI | Requisito de Interface |
| RD | Requisito de Dados |
| API | Application Programming Interface (Interface de Programação de Aplicações) |
| JWT | JSON Web Token (Token Web JSON para autenticação) |
| CRUD | Create, Read, Update, Delete (Operações básicas de dados) |
| DTO | Data Transfer Object (Objeto de Transferência de Dados) |
| UI | User Interface (Interface do Usuário) |
| UX | User Experience (Experiência do Usuário) |
| BPMN | Business Process Model and Notation (Modelo e Notação de Processos de Negócio) |
| UML | Unified Modeling Language (Linguagem de Modelagem Unificada) |
| SQL | Structured Query Language (Linguagem de Consulta Estruturada) |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| JSON | JavaScript Object Notation |

### 1.4 Referências

* Documentação do Spring Boot: https://spring.io/projects/spring-boot
* Documentação do React: https://react.dev/
* Documentação do JWT (java-jwt): https://github.com/auth0/java-jwt
* Padrões de API RESTful.
* Arquivos do projeto ShareCondo (back-end e front-end).
* MODELO DE DOCUMENTAÇÃO DE REQUISITOS.docx

### 1.5 Visão Geral do Documento

Este documento está organizado da seguinte forma:

**Seção 1: Introdução** – Fornece uma visão geral do documento. [cite: 17]

**Seção 2: Descrição Geral** – Apresenta uma visão de alto nível do produto. [cite: 18]

**Seção 3: Requisitos Específicos** – Detalha os requisitos funcionais e não-funcionais. [cite: 19]

**Seção 4: Modelagem de Processos de Negócio (BPMN)** – Descreve processos críticos do sistema.

**Seção 5: Modelagem UML e Diagramas** – Inclui diagramas UML para modelagem do sistema.

**Seção 6: Protótipos e Wireframes** – Apresenta representações visuais da interface.

**Seção 7: Análise de Prioridade de Recursos** – Analisa funcionalidades chave.

**Seção 8: Casos de Teste Funcionais** – Define testes para as funcionalidades.

**Seção 9: Matrizes de Rastreabilidade** – Mapeia requisitos a objetivos e casos de teste. [cite: 20

**Seção 10: Glossário** – Lista termos e abreviações.

**Seção 11: Referências Bibliográficas e Ferramentas** – Cita fontes e ferramentas.

**Apêndices** – Contém informações complementares. [cite: 21]

## 2. DESCRIÇÃO GERAL

### 2.1 Perspectiva do Produto

O ShareCondo é um sistema independente, projetado para ser uma plataforma web completa. [cite: 22] Ele possui uma arquitetura cliente-servidor, com um front-end desenvolvido em React e um back-end em Java com Spring Boot. A comunicação entre o cliente e o servidor ocorre via API RESTful. O sistema utiliza um banco de dados SQL Server para persistência de dados.

### 2.2 Funções do Produto

As principais funções do sistema ShareCondo incluem: [cite: 23]

**Autenticação e Gerenciamento de Perfis:**

* Permitir o cadastro de novos usuários (Moradores, Síndicos, Administradores).
* Autenticar usuários através de login (email e senha).
* Permitir que usuários atualizem seus próprios dados cadastrais.

**Gerenciamento de Condomínios:**

* Permitir que Administradores cadastrem novos condomínios.
* Listar condomínios para seleção durante o cadastro de Moradores e Síndicos.

**Aprovação de Usuários:**

* Permitir que Síndicos aprovem ou rejeitem cadastros de Moradores para seus respectivos condomínios.
* Permitir que Administradores aprovem ou rejeitem quaisquer usuários pendentes.

**Criação e Gerenciamento de Anúncios:**

* Permitir que usuários (aprovados) criem anúncios de itens ou serviços.
* Permitir que usuários visualizem anúncios (filtrados por condomínio, exceto para Admin).
* Permitir que usuários visualizem e gerenciem seus próprios anúncios.

**Criação e Gerenciamento de Ofertas:**

* Permitir que usuários (aprovados) façam ofertas em anúncios de outros usuários (dentro do mesmo condomínio, exceto Admin).
* Permitir que o anunciante aceite ou recuse ofertas recebidas.
* Visualizar ofertas feitas e recebidas.

**Administração do Sistema (Painel Admin):**

* Gerenciar usuários (listar, editar, excluir).
* Gerenciar condomínios (listar, editar, excluir).
* Gerenciar anúncios (listar, editar, excluir).
* Gerenciar ofertas (visualizar).

### 2.3 Características dos Usuários

O sistema será utilizado pelos seguintes tipos de usuários:

* **Morador (USUARIO):**
  + **Nível Educacional:** Variado.
  + **Experiência:** Usuário comum de internet e aplicações web.
  + **Conhecimento Técnico:** Baixo a médio.
  + **Funções Principais:** Cadastrar-se, visualizar anúncios do seu condomínio, criar anúncios, fazer/receber ofertas.
* **Síndico (SINDICO):**
  + **Nível Educacional:** Variado, geralmente com alguma experiência em gestão.
  + **Experiência:** Usuário de internet e aplicações web, possivelmente com alguma experiência em sistemas de gerenciamento.
  + **Conhecimento Técnico:** Médio.
  + **Funções Principais:** Todas as funções de Morador, mais aprovar/rejeitar cadastros de moradores do seu condomínio.
* **Administrador (ADMIN):**
  + **Nível Educacional:** Geralmente com formação técnica ou superior.
  + **Experiência:** Avançada em sistemas web e gerenciamento de plataformas.
  + **Conhecimento Técnico:** Alto.
  + **Funções Principais:** Gerenciamento completo de usuários, condomínios, anúncios e ofertas de toda a plataforma.

### 2.4 Restrições

* O sistema deve ser acessível através de navegadores web modernos.
* A autenticação é obrigatória para a maioria das funcionalidades.
* Usuários do tipo “USUARIO” só podem criar um número limitado de anúncios ativos (MAX\_ACTIVE\_ANNOUNCEMENTS\_PER\_USER = 5).
* A criação de ofertas entre usuários (não ADMIN) é restrita a membros do mesmo condomínio.
* A senha dos usuários deve ser armazenada de forma criptografada (BCrypt).
* O acesso às funcionalidades da API deve ser protegido por JWT e verificação de papéis.

### 2.5 Suposições e Dependências

* **Suposições:**
  + Os usuários possuem acesso à internet e um navegador web.
  + Os usuários fornecerão informações verídicas durante o cadastro.
  + Os síndicos realizarão a aprovação de moradores de forma diligente.
  + A infraestrutura do servidor (onde o back-end e o banco de dados estão hospedados) é estável e disponível.
* **Dependências:**
  + Funcionamento do servidor de banco de dados (SQL Server).
  + Disponibilidade do servidor de aplicação Java (Spring Boot).
  + Bibliotecas de terceiros (listadas no pom.xml e package.json) devem funcionar conforme esperado.

## 3. REQUISITOS ESPECÍFICOS

### 3.1 Requisitos Funcionais

**RF-001: Cadastro de Usuário**

**Descrição:** O sistema deve permitir que um novo usuário se cadastre fornecendo nome, email, senha e, se for do tipo MORADOR ou SINDICO, selecionando um condomínio existente. [cite: 24] \* **Categoria:** Requisito Funcional

**Prioridade:** Alta

**Fonte:** Análise dos arquivos AuthenticationController.java, RegisterDTO.java, CadastroPage.js.

**Critérios de Aceitação:**

1. O usuário deve conseguir submeter o formulário de cadastro com dados válidos.

2. O sistema não deve permitir cadastro com email já existente.

3. A senha deve ser confirmada no formulário.

4. Usuários do tipo MORADOR ou SINDICO devem obrigatoriamente selecionar um condomínio.

5. Após o cadastro, usuários do tipo MORADOR ficam com status “PENDENTE\_APROVACAO”.

6. Usuários do tipo ADMIN e SÍNDICO são cadastrados com status “APROVADO” automaticamente.

**Dependências:** RF-006 (Listagem de Condomínios).

**RF-002: Login de Usuário**

**Descrição:** O sistema deve permitir que um usuário registrado acesse o sistema fornecendo email e senha. [cite: 24]

**Categoria:** Requisito Funcional

**Prioridade:** Alta

**Fonte:** Análise dos arquivos AuthenticationController.java, AuthenticationDTO.java, LoginPage.js.

**Critérios de Aceitação:**

1. O usuário deve conseguir submeter o formulário de login.

2. O sistema deve validar as credenciais.

3. Em caso de sucesso, um token JWT deve ser gerado e retornado ao cliente.

4. Usuários do tipo USUARIO com status diferente de “APROVADO” não devem conseguir logar e devem receber uma mensagem informativa.

5. O sistema deve armazenar os dados do usuário logado (incluindo token) no cliente (localStorage).

**Dependências:** RF-001.

**RF-003: Logout de Usuário**

**Descrição:** O sistema deve permitir que um usuário logado encerre sua sessão. [cite: 24]

**Categoria:** Requisito Funcional

**Prioridade:** Média

**Fonte:** Análise do arquivo Cabecalho/index.js.

**Critérios de Aceitação:** 1. O usuário deve ter a opção de “Sair”. 2. Ao sair, o token e os dados do usuário devem ser removidos do armazenamento local do cliente. 3. O usuário deve ser redirecionado para a página inicial.

**Dependências:** RF-002.

**RF-004: Gerenciamento de Perfil (Básico)**

**Descrição:** O sistema deve permitir que um usuário logado visualize e atualize seus dados cadastrais básicos (nome, email, senha). [cite: 24]

**Categoria:** Requisito Funcional

**Prioridade:** Média

**Fonte:** Análise do arquivo UsuarioController.java (métodos PUT/PATCH).

**Critérios de Aceitação:**

1. O usuário logado pode submeter alterações para seu nome.

2. O usuário logado pode submeter alterações para seu email (validação de unicidade se alterado).

3. O usuário logado pode submeter alteração de senha.

4. ADMIN pode alterar tipo, status e condomínio de outros usuários.

**Dependências:** RF-002.

**RF-005: Criação de Condomínio (Admin)**

**Descrição:** O sistema deve permitir que um usuário ADMIN crie um novo condomínio informando nome e endereço. [cite: 24]

**Categoria:** Requisito Funcional

**Prioridade:** Alta (para funcionalidade do sistema)

**Fonte:** Análise do arquivo CondominioController.java, AddCondoModal.js.

**Critérios de Aceitação:** 1. Um ADMIN logado deve conseguir submeter o formulário de criação de condomínio.

2. O novo condomínio deve ser persistido no banco de dados.

**Dependências:** RF-002 (Login de Admin).

**RF-006: Listagem de Condomínios**

**Descrição:** O sistema deve permitir a listagem de todos os condomínios cadastrados. [cite: 24]

**Categoria:** Requisito Funcional

**Prioridade:** Alta

**Fonte:** Análise do arquivo CondominioController.java, CadastroPage.js (para preencher select).

**Critérios de Aceitação:**

1. Usuários (mesmo não logados, para cadastro) devem conseguir visualizar a lista de condomínios.

2. A lista deve exibir nome e endereço dos condomínios.

**Dependências:** RF-005.

**RF-007: Aprovação/Rejeição de Cadastro de Morador (Síndico/Admin)**

**Descrição:** O sistema deve permitir que um SINDICO aprove ou rejeite o cadastro de moradores pendentes em seu condomínio. ADMINs podem aprovar/rejeitar qualquer usuário pendente. [cite: 24]

**Categoria:** Requisito Funcional

**Prioridade:** Alta

**Fonte:** Análise dos arquivos UsuarioController.java, AprovarUsuariosPage.js.

**Critérios de Aceitação:**

1. SINDICO logado deve visualizar uma lista de usuários pendentes de seu condomínio.

2. ADMIN logado deve visualizar uma lista de todos os usuários pendentes.

3. SINDICO/ADMIN deve conseguir aprovar um cadastro pendente, mudando seu status para “APROVADO”.

4. SINDICO/ADMIN deve conseguir rejeitar um cadastro pendente, mudando seu status para “REJEITADO”.

**Dependências:** RF-001, RF-002.

**RF-008: Criação de Anúncio**

**Descrição:** O sistema deve permitir que um usuário logado e APROVADO crie um anúncio de item ou serviço, fornecendo título, descrição e tipo. [cite: 24]

**Categoria:** Requisito Funcional

**Prioridade:** Alta

**Fonte:** Análise dos arquivos AnuncioController.java, AnuncioRequestDTO.java, AddAnuncioModal.js.

**Critérios de Aceitação:**

1. Usuário logado e APROVADO deve conseguir submeter o formulário de criação de anúncio.

2. O anúncio deve ser associado ao usuário logado.

3. Usuários do tipo USUARIO têm um limite de anúncios ativos (MAX\_ACTIVE\_ANNOUNCEMENTS\_PER\_USER = 5). 4. O anúncio é criado como “ATIVO” por padrão.

**Dependências:** RF-002, (RF-007 ou auto-aprovação para Síndico/Admin).

**RF-009: Listagem de Anúncios**

**Descrição:** O sistema deve listar anúncios ativos. Usuários comuns e síndicos veem apenas anúncios de seu condomínio. ADMINs veem todos. [cite: 24]

**Categoria:** Requisito Funcional

**Prioridade:** Alta

**Fonte:** Análise dos arquivos AnuncioController.java, AnunciosPage.js.

**Critérios de Aceitação:**

1. Usuários logados e APROVADOS podem ver a lista de anúncios.

2. ADMINs veem todos os anúncios ativos.

3. SINDICO e USUARIO veem apenas anúncios ativos de usuários de seu próprio condomínio.

**Dependências:** RF-008.

**RF-010: Visualização de Detalhes do Anúncio**

**Descrição:** O sistema deve permitir que um usuário logado e APROVADO visualize os detalhes completos de um anúncio. [cite: 24]

**Categoria:** Requisito Funcional

**Prioridade:** Alta

**Fonte:** Análise dos arquivos AnuncioController.java (método buscarAnuncioPorId), AnuncioDetalhePage.js.

**Critérios de Aceitação:**

1. Usuários logados e APROVADOS podem acessar a página de detalhes de um anúncio.

2. A visualização é restrita ao condomínio do usuário, exceto para ADMIN.

3. Todos os dados relevantes do anúncio (título, descrição, tipo, anunciante, data, status) são exibidos.

**Dependências:** RF-009.

**RF-011: Gerenciamento dos Próprios Anúncios**

**Descrição:** O sistema deve permitir que um usuário logado e APROVADO visualize, edite e delete seus próprios anúncios. [cite: 24]

**Categoria:** Requisito Funcional

**Prioridade:** Média

**Fonte:** Análise dos arquivos AnuncioController.java (métodos /meus, PUT, DELETE), MeusAnunciosPage.js.

**Critérios de Aceitação:**

1. Usuário pode listar todos os seus anúncios (ativos e inativos).

2. Usuário pode editar título, descrição, tipo e status (ativo/inativo) de seus anúncios. 3. Usuário pode excluir seus anúncios.

**Dependências:** RF-008.

**RF-012: Criação de Oferta**

**Descrição:** O sistema deve permitir que um usuário logado e APROVADO faça uma oferta em um anúncio ativo de outro usuário (do mesmo condomínio, exceto se um dos envolvidos for ADMIN). [cite: 24]

**Categoria:** Requisito Funcional

**Prioridade:** Alta

**Fonte:** Análise dos arquivos OfertaController.java, OfertaRequestDTO.java, OfferTradeModal.js.

**Critérios de Aceitação:**

1. Usuário logado e APROVADO pode submeter uma oferta.

2. Não é possível ofertar no próprio anúncio.

3. Ofertas entre USUARIOs/SINDICO são restritas ao mesmo condomínio (a menos que um seja ADMIN).

4. A oferta pode ser do tipo DINHEIRO (com valor) ou ITEM/SERVICO (com descrição).

5. A oferta é criada com status “PENDENTE”.

**Dependências:** RF-009.

**RF-013: Aceitar Oferta**

**Descrição**: O sistema deve permitir que o anunciante aceite uma oferta PENDENTE para seu anúncio ativo. [cite: 24] Categoria: Requisito Funcional

**Prioridade:** Alta

**Fonte:** Análise dos arquivos OfertaController.java, MeusAnunciosPage.js.

**Critérios de Aceitação:**

1. Apenas o anunciante do anúncio pode aceitar uma oferta.

2. A oferta deve estar com status “PENDENTE”.

3. O anúncio deve estar “ATIVO”.

4. Ao aceitar, o status da oferta muda para “ACEITA”.

5. O anúncio relacionado é desativado (status “INATIVO”).

6. Outras ofertas PENDENTES para o mesmo anúncio são automaticamente RECUSADAS.

**Dependências**: RF-012.

**RF-014: Recusar Oferta**

**Descrição**: O sistema deve permitir que o anunciante recuse uma oferta PENDENTE para seu anúncio. [cite: 24]

**Categoria:** Requisito Funcional

**Prioridade**: Média

**Fonte:** Análise dos arquivos OfertaController.java, MeusAnunciosPage.js.

**Critérios de Aceitação:**

1. Apenas o anunciante do anúncio pode recusar uma oferta.

2. A oferta deve estar com status “PENDENTE”.

3. Ao recusar, o status da oferta muda para “RECUSADA”.

**Dependências:** RF-012.

**RF-015: Visualização de Minhas Ofertas (Feitas)**

**Descrição:** O sistema deve permitir que um usuário logado e APROVADO visualize as ofertas que ele fez em anúncios de outros. [cite: 24]

**Categoria**: Requisito Funcional

**Prioridade**: Média

**Fonte:** Análise dos arquivos OfertaController.java (método listarOfertasPorUsuario), MinhasOfertasPage.js.

**Critérios de Aceitação:**

1. Usuário logado e APROVADO pode ver a lista de suas ofertas enviadas.

2. A lista deve exibir detalhes da oferta (anúncio alvo, tipo, valor/descrição, data, status).

### Dependências: RF-012.

### RF-016: Gerenciamento de Entidades (Admin)

### Descrição: Usuários ADMIN devem ter acesso a funcionalidades CRUD completas para Usuários, Condomínios e Anúncios. Para Ofertas, a funcionalidade é primariamente de visualização. [cite: 24]

### Categoria: Requisito Funcional

### Prioridade: Alta

### Fonte: Análise dos arquivos UsuarioController.java, CondominioController.java, AnuncioController.java, OfertaController.java, e páginas de Admin (UserManagementPage.js, CondoManagementPage.js, AnuncioManagementPage.js, OfertaManagementPage.js).

### Critérios de Aceitação:

### 1. ADMIN pode criar, listar, visualizar por ID, atualizar e excluir qualquer Usuário.

### 2. ADMIN pode criar, listar, visualizar por ID, atualizar e excluir qualquer Condomínio.

### 3. ADMIN pode criar, listar, visualizar por ID, atualizar e excluir qualquer Anúncio.

### 4. ADMIN pode listar e visualizar ofertas (filtrando por anúncio ou usuário).

### Dependências: RF-002.

### 3.2 Requisitos Não-Funcionais

### RNF-001: Usabilidade - Interface Intuitiva

### Descrição: A interface do usuário deve ser clara, intuitiva e fácil de usar para todos os tipos de usuários, minimizando a curva de aprendizado. [cite: 25]

### Categoria: Requisito Não-Funcional (Usabilidade)

### Prioridade: Alta

### Fonte: Boas práticas de UX/UI.

### Critérios de Aceitação:

### 1. Novos usuários conseguem se cadastrar e criar um anúncio em menos de 5 minutos sem necessidade de manual.

### 2. As principais funcionalidades devem ser acessíveis com no máximo 3 cliques a partir da página principal (após login).

### 3. Feedback visual claro (mensagens de sucesso, erro, indicadores de carregamento) deve ser fornecido para todas as ações do usuário.

### Dependências: N/A.

### RNF-002: Desempenho - Tempo de Resposta da API

### Descrição: As requisições à API devem responder em tempo hábil para não prejudicar a experiência do usuário. [cite: 25]

### Categoria: Requisito Não-Funcional (Desempenho)

### Prioridade: Alta

### Fonte: Necessidade de boa UX.

### Critérios de Aceitação:

### 1. 95% das requisições de leitura (GET) da API devem responder em menos de 500ms sob carga de até 50 usuários simultâneos.

### 2. 95% das requisições de escrita (POST, PUT, PATCH, DELETE) da API devem responder em menos de 1000ms sob carga de até 50 usuários simultâneos.

### Dependências: N/A.

### RNF-003: Segurança - Proteção de Dados do Usuário

### Descrição: Os dados dos usuários, especialmente senhas e informações pessoais, devem ser protegidos contra acesso não autorizado. [cite: 25]

### Categoria: Requisito Não-Funcional (Segurança)

### Prioridade: Alta

### Fonte: LGPD, Boas práticas de segurança.

### Critérios de Aceitação:

### 1. Senhas devem ser armazenadas usando hashing BCrypt.

### 2. A comunicação entre cliente e servidor deve utilizar HTTPS em ambiente de produção.

### 3. Acesso às funcionalidades da API deve ser protegido por autenticação (JWT) e autorização baseada em papéis (ROLE\_ADMIN, ROLE\_SINDICO, ROLE\_USER).

### 4. Tokens JWT devem ter tempo de expiração configurado (ex: 1 hora, conforme TokenService.java).

### Dependências: N/A.

### RNF-004: Segurança - Validação de Entrada

### Descrição: Todas as entradas de dados, tanto no front-end quanto no back-end, devem ser validadas para prevenir ataques (ex: XSS, SQL Injection) e garantir a integridade dos dados. [cite: 25]

### Categoria: Requisito Não-Funcional (Segurança)

### Prioridade: Alta

### Fonte: Boas práticas de segurança.

### Critérios de Aceitação:

### 1. Dados de entrada nas APIs são validados quanto ao tipo, formato, tamanho e obrigatoriedade (conforme DTOs e lógica nos controllers).

### 2. Entradas de texto que serão exibidas na UI são tratadas para prevenir Cross-Site Scripting (XSS) (responsabilidade do front-end ao renderizar).

### 3. Consultas ao banco de dados utilizam Prepared Statements ou ORM (JPA) que previne SQL Injection.

### Dependências: N/A.

### RNF-005: Disponibilidade

### Descrição: O sistema deve estar disponível para os usuários a maior parte do tempo. [cite: 25]

### Categoria: Requisito Não-Funcional (Disponibilidade)

### Prioridade: Média

### Fonte: Expectativa do usuário.

### Critérios de Aceitação:

### 1. O sistema deve ter um uptime de 99% (excluindo janelas de manutenção programadas e comunicadas com antecedência).

### Dependências: Infraestrutura de hospedagem.

### RNF-006: Manutenibilidade

### Descrição: O código do sistema deve ser bem estruturado, comentado e seguir boas práticas de desenvolvimento para facilitar a manutenção e futuras evoluções. [cite: 25]

### Categoria: Requisito Não-Funcional (Manutenibilidade)

### Prioridade: Média

### Fonte: Necessidades de desenvolvimento.

### Critérios de Aceitação:

### 1. O código segue os padrões de nomenclatura e estilo definidos pela equipe (ex: convenções Java, padrões React).

### 2. Componentes e módulos são razoavelmente desacoplados, promovendo a reutilização (observável na estrutura de componentes React e serviços/controllers no back-end).

### 3. Cobertura de testes unitários no back-end de pelo menos 60% (o arquivo ShareCondoApplicationTests.java existe, mas precisa de expansão).

### 4. Comentários explicativos em trechos complexos do código.

### Dependências: N/A.

### RNF-007: Compatibilidade de Navegadores

### Descrição: A interface front-end deve ser compatível com as versões mais recentes dos principais navegadores (Chrome, Firefox, Safari, Edge). [cite: 25]

### Categoria: Requisito Não-Funcional (Compatibilidade)

### Prioridade: Média

### Fonte: Lista de browsers suportados (browserslist no package.json).

### Critérios de Aceitação:

### 1. As funcionalidades principais são testadas e funcionam corretamente nas duas últimas versões estáveis dos navegadores Chrome, Firefox, Safari e Edge.

### Dependências: N/A.

### RNF-008: Escalabilidade

### Descrição: O sistema deve ser capaz de lidar com um aumento no número de usuários, condomínios, anúncios e ofertas sem degradação significativa de desempenho. [cite: 25]

### Categoria: Requisito Não-Funcional (Escalabilidade)

### Prioridade: Média

### Fonte: Previsão de crescimento do sistema.

### Critérios de Aceitação:

### 1. A arquitetura do back-end (Spring Boot) permite escalonamento horizontal.

### 2. O banco de dados (SQL Server) suporta otimizações e indexações para lidar com aumento de volume de dados.

### Dependências: Infraestrutura de hospedagem.

### 3.3 Requisitos de Interface

### RI-001: Interface Web do Usuário (Front-end React)

### Descrição: O sistema deve prover uma interface gráfica web responsiva para interação dos usuários (Moradores, Síndicos) e Administradores. [cite: 26]

### Tipo de Interface: Usuário [cite: 26]

### Prioridade: Alta

### Fonte: Arquitetura do projeto (React).

### Especificações:

### 1. Página Inicial (HomePage.js): Apresentação do ShareCondo, seções “Como Funciona”, “Benefícios” e CTAs. Layout conforme HomePage.css.

### 2. Página de Cadastro (CadastroPage.js): Formulário para registro de novos usuários, incluindo seleção de condomínio. Layout conforme Formulario.css.

### 3. Página de Login (LoginPage.js): Formulário para autenticação de usuários. Layout conforme Formulario.css.

### 4. Página de Anúncios (AnunciosPage.js): Listagem de anúncios, filtros (implícito), botão para criar novo anúncio. Layout conforme Anuncio.css.

### 5. Página de Detalhes do Anúncio (AnuncioDetalhePage.js): Exibição completa dos dados de um anúncio, opção de fazer oferta. Layout conforme AnuncioDetalhe.css.

### 6. Página “Meus Anúncios” (MeusAnunciosPage.js): Listagem dos anúncios criados pelo usuário logado, com opções de gerenciamento de ofertas recebidas. Layout conforme MeusAnuncios.css.

### 7. Página “Minhas Ofertas” (MinhasOfertasPage.js): Listagem das ofertas feitas pelo usuário logado e seus status. Layout conforme MinhasOfertas.css.

### 8. Página “Aprovar Usuários” (Síndico) (AprovarUsuariosPage.js): Interface para síndicos visualizarem e gerenciarem usuários pendentes de aprovação em seu condomínio. Layout conforme AprovarUsuarios.css.

### 9. Painel Administrativo (AdminLayout.js e subpáginas): Interface para administradores gerenciarem usuários, condomínios, anúncios e ofertas. Layouts conforme AdminLayout.css, CrudTabela.css, AdminModal.css.

### 10. Componentes Reutilizáveis:

### Cabeçalho (Cabecalho/index.js, Cabecalho.css): Logo, links de navegação, informações do usuário, botão de logout/login/cadastro.

### Rodapé (Rodape/Rodape.js, Rodape.css): Informações de copyright, links para redes sociais.

### Botões (Botao/Button.js, Button.css): Componente de botão estilizado e reutilizável.

### Modais (Modal/Modal.js, Modal.css; Anuncios/AddAnuncioModal.js; Anuncios/OfferTradeModal.js; Admin/AddCondoModal.js; Admin/EditUserModal.js): Janelas de diálogo para ações específicas.

### Formulários (Formulario/index.js, Formulario.css; FormInput/index.js, FormInput.css): Estrutura e campos de formulário reutilizáveis.

### Tabelas CRUD (CrudTabela/index.js, CrudTabela.css): Componente para exibir e gerenciar dados em formato de tabela.

### 11. Página “Quem Somos” (QuemSomosPage.js, QuemSomosPage.css): Página informativa sobre o projeto.

### 12. Página “Não Encontrado” (NotFoundPage.js): Página para rotas inválidas.

### Dependências: RNF-001, RNF- 007.

### RI-002: API RESTful do Back-end (Spring Boot)

### Descrição: O sistema deve expor uma API RESTful para comunicação entre o front-end e o back-end, gerenciando todas as operações de negócio. [cite: 26]

### Tipo de Interface: Software (API) [cite: 26]

### Prioridade: Alta

### Fonte: Arquitetura do projeto (Spring Boot).

### Especificações:

### 1. Endpoints de Autenticação (/auth - AuthenticationController.java):

### POST /login: Autentica usuário e retorna JWT.

### POST /register: Registra novo usuário.

### 2. Endpoints de Usuário (/usuario - UsuarioController.java):

### GET /: Lista todos os usuários (Admin).

### GET /{id}: Busca usuário por ID (Autenticado).

### PUT /{id}: Atualiza usuário (Admin ou proprietário).

### PATCH /{id}: Atualiza parcialmente usuário (Admin ou proprietário).

### DELETE /{id}: Deleta usuário (Admin).

### GET /pendentes: Lista usuários pendentes de aprovação (Síndico/Admin).

### PATCH /{id}/aprovar: Aprova usuário (Síndico/Admin).

### PATCH /{id}/rejeitar: Rejeita usuário (Síndico/Admin).

### 3. Endpoints de Condomínio (/condominio - CondominioController.java):

### POST /: Cria condomínio (Admin).

### GET /: Lista todos os condomínios (PermitAll - para cadastro).

### GET /{id}: Busca condomínio por ID (PermitAll).

### PUT /{id}: Atualiza condomínio (Admin).

### DELETE /{id}: Deleta condomínio (Admin).

### 4. Endpoints de Anúncio (/anuncios - AnuncioController.java):

### POST /: Cria anúncio (Usuário aprovado).

### GET /: Lista anúncios (filtrado por condomínio, exceto Admin).

### GET /meus: Lista anúncios do usuário logado.

### GET /{id}: Busca anúncio por ID (restrições de condomínio aplicáveis).

### PUT /{id}: Atualiza anúncio (Anunciante ou Admin).

### DELETE /{id}: Deleta anúncio (Anunciante ou Admin).

### 5. Endpoints de Oferta (/ofertas - OfertaController.java):

### POST /anuncio/{anuncioId}: Cria oferta para um anúncio (Usuário aprovado).

### GET /anuncio/{anuncioId}: Lista ofertas de um anúncio (restrições de condomínio).

### GET /usuario/{usuarioId}: Lista ofertas feitas por um usuário (Proprietário ou Admin/Síndico com restrições).

### PATCH /{ofertaId}/aceitar: Anunciante aceita oferta.

### PATCH /{ofertaId}/recusar: Anunciante recusa oferta.

### GET /: Lista todas as ofertas (Admin/Síndico com restrições).

### 6. Formato de Dados: JSON.

### 7. Autenticação: JWT Bearer Token no header Authorization.

### 8. CORS: Configurado em CorsConfig.java para permitir requisições de http://localhost:3000 com métodos GET, POST, PUT, DELETE, PATCH e todos os headers.

### Dependências: RNF-002, RNF-003, RNF- 004.

### 3.4 Requisitos de Dados

**RD-001:** Dados do Usuário

**Descrição:** Informações sobre os usuários cadastrados no sistema. [cite: 27]

**Prioridade:** Alta

**Fonte**: Usuario.java, UsuarioDTO.java, RegisterDTO.java.

**Especificações:**

Tipo de dados: ID (String UUID), Nome (String), Email (String, único), Senha (String, criptografada), TipoUsuario (Enum: ADMIN, SINDICO, USUARIO), StatusUsuario (Enum: PENDENTE\_APROVACAO, APROVADO, REJEITADO), Condominio (FK para Condominio).

Volume: Estimado em 1000-10000 usuários inicialmente, com crescimento.

Frequência de acesso: Alta (login, visualização de perfis, associação a anúncios/ofertas).

Requisitos de retenção: Indefinido (manter enquanto a conta existir, sujeito a políticas de privacidade).

Requisitos de backup: Backup diário do banco de dados.

**Dependências:** RD-002 (Condominio).

**RD-002: Dados do Condomínio**

**Descrição**: Informações sobre os condomínios cadastrados. [cite: 27]

**Prioridade:** Alta

**Fonte:** Condominio.java, CondominioDTO.java.

**Especificações:**

Tipo de dados: ID (String UUID), Nome (String), Endereco (String).

Volume: Estimado em 10-100 condomínios inicialmente, com crescimento moderado.

Frequência de acesso: Média (cadastro de usuários, listagem, filtros de anúncios).

Requisitos de retenção: Indefinido (manter enquanto o condomínio existir na plataforma).

Requisitos de backup: Backup diário do banco de dados.

**Dependências:** N/A.

**RD-003: Dados do Anúncio**

**Descrição:** Informações sobre os anúncios de itens ou serviços criados pelos usuários. [cite: 27]

**Prioridade:** Alta

**Fonte**: Anuncio.java, AnuncioRequestDTO.java, AnuncioResponseDTO.java.

**Especificações:**

Tipo de dados: ID (String UUID), Titulo (String), Descricao (String), TipoAnuncio (Enum: ITEM, SERVICO), Anunciante (FK para Usuario), DataCriacao (LocalDateTime), Ativo (boolean).

Volume: Estimado em 5-20 anúncios por usuário ativo.

Frequência de acesso: Alta (listagem, visualização de detalhes).

Requisitos de retenção: Manter enquanto o anúncio for relevante ou até ser excluído pelo usuário/admin. Anúncios inativos (após oferta aceita) devem ser mantidos para histórico.

Requisitos de backup: Backup diário do banco de dados.

**Dependências:** RD-001 (Usuario).

**RD-004: Dados da Oferta**

**Descrição:** Informações sobre as ofertas feitas pelos usuários em anúncios. [cite: 27]

**Prioridade:** Alta

**Fonte**: Oferta.java, OfertaRequestDTO.java, OfertaResponseDTO.java.

**Especificações:**

Tipo de dados: ID (String UUID), Anuncio (FK para Anuncio), Ofertante (FK para Usuario), TipoOferta (Enum: DINHEIRO, ITEM, SERVICO), Valor (BigDecimal, opcional), Descricao (String, opcional para item/serviço), DataOferta (LocalDateTime), Status (Enum: PENDENTE, ACEITA, RECUSADA, CANCELADA).

Volume: Estimado em 0-10 ofertas por anúncio.

Frequência de acesso: Média a Alta (criação, visualização pelo anunciante e ofertante).

Requisitos de retenção: Manter enquanto o anúncio e as contas dos usuários envolvidos existirem.

Requisitos de backup: Backup diário do banco de dados.

**Dependências:** RD-001 (Usuario), RD-003 (Anuncio).

## 4. MODELAGEM DE PROCESSOS DE NEGÓCIO (BPMN)

**[ESPAÇO RESERVADO PARA DIAGRAMA BPMN COLABORATIVO: CADASTRO E APROVAÇÃO DE NOVO MORADOR]**

**(Por favor, insira aqui o diagrama BPMN gerado por uma ferramenta apropriada, seguindo a descrição textual abaixo).**

**Descrição do Processo (Cadastro e Aprovação de Novo Morador):**

* **Pools:** “Candidato a Morador”, “Sistema ShareCondo”, “Síndico do Condomínio”.
* **Lane: Candidato a Morador**
  1. **Evento de Início:** Deseja se cadastrar.
  2. **Tarefa:** Acessa página de cadastro.
  3. **Tarefa:** Preenche formulário de cadastro (nome, email, senha, seleciona condomínio).
  4. **Tarefa:** Submete formulário.
  5. **Evento Intermediário de Mensagem (Envio):** Dados de cadastro enviados.
  6. **Evento Intermediário de Tempo/Condição:** Aguarda aprovação.
  7. **Gateway Exclusivo (Baseado em Evento):** Cadastro Aprovado OU Cadastro Rejeitado?
     + **Se Aprovado:** **Evento de Fim:** Acesso concedido.
     + **Se Rejeitado:** **Evento de Fim:** Cadastro negado.
* **Lane: Sistema ShareCondo**
  1. **Evento Intermediário de Mensagem (Recebimento):** Recebe dados de cadastro.
  2. **Subprocesso:** Validar Dados do Cadastro.
     + **Tarefa:** Verificar se email já existe.
     + **Tarefa:** Validar formato dos dados (senha, email).
     + **Gateway Exclusivo:** Dados válidos?
       - **Se Não:** **Tarefa:** Exibe mensagem de erro ao Candidato. **Fluxo termina aqui para o sistema neste ponto (usuário corrige ou desiste).**
       - **Se Sim:** Continua.
  3. **Tarefa:** Criar registro de usuário com status “PENDENTE\_APROVACAO”.
  4. **Tarefa:** Associar usuário ao condomínio selecionado.
  5. **Tarefa:** Notificar Síndico sobre novo cadastro pendente (ex: via sistema/email - funcionalidade de notificação não detalhada no escopo atual, mas implícita para o fluxo).
  6. **Evento Intermediário de Mensagem (Envio):** Notificação enviada ao Síndico.
  7. **(Após decisão do Síndico)**
     + **Se Aprovado pelo Síndico:** **Tarefa:** Atualiza status do usuário para “APROVADO”. **Tarefa:** Notifica Morador sobre aprovação.
     + **Se Rejeitado pelo Síndico:** **Tarefa:** Atualiza status do usuário para “REJEITADO”. **Tarefa:** Notifica Morador sobre rejeição.
* **Lane: Síndico do Condomínio**
  1. **Evento Intermediário de Mensagem (Recebimento):** Recebe notificação de novo cadastro.
  2. **Tarefa:** Acessa painel de aprovação de usuários.
  3. **Tarefa:** Analisa dados do candidato.
  4. **Gateway Exclusivo:** Aprovar cadastro?
     + **Se Sim:** **Tarefa:** Clica em “Aprovar” no sistema. **Evento Intermediário de Mensagem (Envio):** Decisão de aprovação enviada ao Sistema.
     + **Se Não:** **Tarefa:** Clica em “Rejeitar” no sistema. **Evento Intermediário de Mensagem (Envio):** Decisão de rejeição enviada ao Sistema.
  5. **Evento de Fim:** Processo de aprovação/rejeição concluído pelo Síndico.

## 5. MODELAGEM UML E DIAGRAMAS

### 5.1. Modelo de Domínio (Diagrama de Classes de Alto Nível)

**[ESPAÇO RESERVADO PARA DIAGRAMA DE CLASSES DE ALTO NÍVEL (MODELO DE DOMÍNIO)]**

**(Por favor, insira aqui o diagrama de classes de alto nível gerado por uma ferramenta apropriada, com base na descrição abaixo).**

* **Descrição:** Este diagrama representaria as principais entidades do sistema e seus relacionamentos.
* **Classes Principais:**
  + Usuario: Atributos chave como id, nome, email, tipoUsuario, statusUsuario.
  + Condominio: Atributos chave como id, nome, endereco.
  + Anuncio: Atributos chave como id, titulo, descricao, tipoAnuncio, ativo, dataCriacao.
  + Oferta: Atributos chave como id, tipoOferta, valor, descricao, dataOferta, status.
* **Relacionamentos Principais:**
  + Usuario (1) — (0..1) Condominio (Um usuário (MORADOR/SINDICO) está associado a um condomínio; um ADMIN pode não estar. Um condomínio pode ter muitos usuários).
  + Usuario (Anunciante) (1) — (0..\*) Anuncio (Um usuário cria zero ou muitos anúncios).
  + Anuncio (1) — (0..\*) Oferta (Um anúncio recebe zero ou muitas ofertas).
  + Usuario (Ofertante) (1) — (0..\*) Oferta (Um usuário faz zero ou muitas ofertas).

### 5.2. Diagrama de Casos de Uso

**[ESPAÇO RESERVADO PARA DIAGRAMA DE CASOS DE USO]**

**(Por favor, insira aqui o diagrama de casos de uso gerado por uma ferramenta apropriada, com base na descrição abaixo).**

* **Descrição:** Este diagrama mostraria os atores do sistema e os principais casos de uso que eles podem realizar.
* **Atores:**
  + Morador (Generaliza para Usuário comum)
  + Síndico (Especialização de Morador)
  + Administrador
  + Visitante (Usuário não autenticado)
* **Casos de Uso Principais:**
  + Manter Cadastro (Inclui: Cadastrar, Visualizar Perfil, Editar Perfil) - Atores: Morador, Síndico, Administrador.
  + Realizar Login/Logout - Atores: Morador, Síndico, Administrador, (Visitante para Login).
  + Gerenciar Condomínios (Inclui: Criar, Listar, Editar, Excluir Condomínio) - Ator: Administrador.
  + Visualizar Condomínios - Atores: Visitante, Morador, Síndico, Administrador.
  + Aprovar/Rejeitar Morador - Atores: Síndico, Administrador.
  + Gerenciar Anúncios (Inclui: Criar, Listar Próprios, Editar Próprio, Excluir Próprio Anúncio) - Atores: Morador (aprovado), Síndico, Administrador (CRUD completo).
  + Visualizar Anúncios (Públicos/Condomínio) - Atores: Morador (aprovado), Síndico, Administrador.
  + Gerenciar Ofertas (Inclui: Fazer Oferta, Visualizar Ofertas Feitas, Visualizar Ofertas Recebidas, Aceitar Oferta, Recusar Oferta) - Atores: Morador (aprovado), Síndico, Administrador (visualização expandida).

### 5.3. Diagrama de Sequência (para o Caso de Uso “Criar Anúncio”)

**[ESPAÇO RESERVADO PARA DIAGRAMA DE SEQUÊNCIA: CRIAR ANÚNCIO]**

**(Por favor, insira aqui o diagrama de sequência gerado por uma ferramenta apropriada, com base na descrição abaixo).**

* **Descrição:** Este diagrama detalharia a sequência de interações entre objetos para o caso de uso “Criar Anúncio”.
* **Objetos/Participantes:**
  + :UsuarioAtor (Front-end/Usuário)
  + :AnunciosPage (Componente React)
  + :AddAnuncioModal (Componente React)
  + anunciosServico:anuncios.js (Serviço Front-end)
  + anunciosAPI:axios (Cliente HTTP)
  + :AnuncioController (Back-end)
  + :UsuarioRepository (Back-end)
  + :AnuncioRepository (Back-end)
  + novoAnuncio:Anuncio (Entidade Back-end)
* **Sequência Principal:**
  1. UsuarioAtor clica em “Adicionar Novo Anúncio” em AnunciosPage.
  2. AnunciosPage abre AddAnuncioModal.
  3. UsuarioAtor preenche dados (título, descrição, tipo) em AddAnuncioModal e submete.
  4. AddAnuncioModal chama onAddAnuncio (prop) com os dados.
  5. AnunciosPage (handler handleAddAnuncio) forma o payload e chama createAnuncio(payload) em anunciosServico.
  6. anunciosServico faz uma requisição POST para /anuncios via anunciosAPI.
  7. AnuncioController recebe a requisição.
  8. AnuncioController obtém o Usuario anunciante via SecurityContextHolder e UsuarioRepository.
  9. AnuncioController valida o status do usuário e o limite de anúncios.
  10. AnuncioController cria uma instância de novoAnuncio:Anuncio com os dados.
  11. AnuncioController chama save(novoAnuncio) em AnuncioRepository.
  12. AnuncioRepository persiste o anúncio.
  13. AnuncioController retorna uma resposta HTTP 201 (Created) com o AnuncioResponseDTO.
  14. anunciosAPI recebe a resposta.
  15. anunciosServico retorna os dados para AnunciosPage.
  16. AnunciosPage atualiza o estado (feedback, recarrega lista de anúncios), fecha o modal.

### 5.4. Diagrama de Classes Detalhado

**[ESPAÇO RESERVADO PARA DIAGRAMA DE CLASSES DETALHADO]**

**(Por favor, insira aqui o diagrama de classes detalhado gerado por uma ferramenta apropriada, com base na descrição abaixo).**

* **Descrição:** Este diagrama expandiria o Modelo de Domínio, mostrando atributos com tipos, métodos principais das classes, e detalhando mais os relacionamentos e enumerações.
* **Classes (com mais detalhes):**
  + Usuario: id: String, nome: String, email: String, senha: String, tipoUsuario: TipoUsuario, condominio: Condominio, statusUsuario: StatusUsuario. Métodos como getAuthorities(), isEnabled(), construtores, getters/setters.
  + Condominio: id: String, nome: String, endereco: String. Getters/setters.
  + Anuncio: id: String, titulo: String, descricao: String, tipoAnuncio: TipoAnuncio, anunciante: Usuario, dataCriacao: LocalDateTime, ativo: boolean, ofertasRecebidas: List<Oferta>. Getters/setters.
  + Oferta: id: String, anuncio: Anuncio, ofertante: Usuario, tipoOferta: TipoOferta, valor: BigDecimal, descricao: String, dataOferta: LocalDateTime, status: StatusOferta. Getters/setters.
  + **Enums:** TipoUsuario, StatusUsuario, TipoAnuncio, TipoOferta, StatusOferta.
  + **DTOs:** UsuarioDTO, CondominioDTO, AnuncioRequestDTO, AnuncioResponseDTO, OfertaRequestDTO, OfertaResponseDTO, RegisterDTO, AuthenticationDTO, LoginResponseDTO, UsuarioUpdateRequestDTO. Mostrar seus atributos (records são concisos).
  + **Controllers:** AuthenticationController, UsuarioController, CondominioController, AnuncioController, OfertaController. Mostrar os principais métodos de endpoint e suas anotações (ex: @PostMapping, @GetMapping).
  + **Repositories:** UsuarioRepository, CondominioRepository, AnuncioRepository, OfertaRepository. Mostrar que estendem JpaRepository.
  + **Services/Infra:** TokenService, SecurityFilter, SecurityConfiguration, AuthorizationService, CorsConfig. Mostrar dependências principais.
* **Relacionamentos:**
  + Detalhar cardinalidades e tipos de associação (ex: @ManyToOne, @OneToMany).

## 6. PROTÓTIPOS E WIREFRAMES

### 6.1. Wireframes

As telas do front-end desenvolvidas em React, localizadas em share-condo-front-end/src/paginas/, servem como protótipos funcionais de média fidelidade para as principais interações do usuário. Cada subdiretório em paginas (e.g., Login, Cadastro, Anuncios, Admin) contém os componentes que formam as telas do sistema.

**Exemplos de Telas Principais (Arquivos de Referência):** \* **Login:** share-condo-front-end/src/paginas/Login/LoginPage.js \* **Cadastro:** share-condo-front-end/src/paginas/Cadastro/CadastroPage.js \* **Listagem de Anúncios:** share-condo-front-end/src/paginas/Anuncios/AnunciosPage.js \* **Detalhes do Anúncio:** share-condo-front-end/src/paginas/AnuncioDetalhe/AnuncioDetalhePage.js \* **Meus Anúncios:** share-condo-front-end/src/paginas/MeusAnuncios/MeusAnunciosPage.js \* **Minhas Ofertas:** share-condo-front-end/src/paginas/MinhasOfertas/MinhasOfertasPage.js \* **Aprovação de Usuários (Síndico):** share-condo-front-end/src/paginas/Sindico/AprovarUsuariosPage.js \* **Painel Admin (Ex: Gerenciamento de Usuários):** share-condo-front-end/src/paginas/Admin/UserManagementPage.js

**[ESPAÇO RESERVADO PARA IMAGENS DE WIREFRAMES DAS PRINCIPAIS TELAS, SE NECESSÁRIO ADICIONALMENTE AOS COMPONENTES REACT EXISTENTES. CONSIDERE CAPTURAS DE TELA DAS PÁGINAS EM EXECUÇÃO.]**

### 6.2. Storyboard: “Usuário Cria Anúncio e Outro Usuário Faz uma Oferta”

1. **Cena 1: Usuário A (Anunciante) Decide Anunciar**
   * **Ação:** Usuário A, logado e aprovado, navega para a página “Anúncios”.
   * **Tela:** AnunciosPage.js exibindo anúncios existentes.
   * **Usuário:** Clica no botão “+ Adicionar Novo Anúncio”.
2. **Cena 2: Usuário A Preenche Dados do Anúncio**
   * **Ação:** Um modal (AddAnuncioModal.js) é exibido.
   * **Tela:** Modal com campos para Título, Descrição e Tipo de Anúncio.
   * **Usuário:** Preenche os detalhes do item que deseja anunciar (ex: “Furadeira usada, bom estado”, tipo “ITEM”) e clica em “Adicionar Anúncio”.
3. **Cena 3: Anúncio Publicado**
   * **Ação:** O sistema processa a criação do anúncio.
   * **Tela:** AnunciosPage.js é atualizada, mostrando o novo anúncio do Usuário A na lista. Mensagem de sucesso é exibida.
4. **Cena 4: Usuário B (Interessado) Visualiza Anúncios**
   * **Ação:** Usuário B (do mesmo condomínio, logado e aprovado) navega para a página “Anúncios”.
   * **Tela:** AnunciosPage.js exibindo anúncios, incluindo o do Usuário A.
   * **Usuário:** Vê o anúncio da furadeira e clica em “Ver Detalhes” ou diretamente em “Fazer Oferta”.
5. **Cena 5: Usuário B Faz uma Oferta**
   * **Ação:** Se clicou em “Fazer Oferta”, o modal OfferTradeModal.js é aberto (ou após ver detalhes em AnuncioDetalhePage.js e clicar em Fazer Oferta).
   * **Tela:** Modal para fazer oferta, mostrando o título do anúncio da furadeira.
   * **Usuário:** Seleciona o tipo de oferta (ex: “DINHEIRO”), preenche o valor (ex: R$ 30,00) e clica em “Enviar Oferta”.
6. **Cena 6: Oferta Enviada**
   * **Ação:** Sistema processa a criação da oferta.
   * **Tela:** Modal fecha. Mensagem de sucesso para Usuário B.
7. **Cena 7: Usuário A (Anunciante) Vê a Oferta**
   * **Ação:** Usuário A navega para “Meus Anúncios”.
   * **Tela:** MeusAnunciosPage.js lista seus anúncios. O anúncio da furadeira mostra que há uma nova oferta pendente.
   * **Usuário:** Expande as ofertas do anúncio da furadeira e vê a oferta de R$ 30,00 do Usuário B.

**[ESPAÇO RESERVADO PARA ILUSTRAÇÕES DO STORYBOARD, SE DESEJADO.]**

### 6.3. Story Map

**[ESPAÇO RESERVADO PARA UM STORY MAP VISUAL. A DESCRIÇÃO ABAIXO PODE SER USADA COMO BASE PARA CRIÁ-LO.]**

**Épico 1: Gerenciamento de Contas de Usuário**

Atividades/Features (Release 1 - Prioridade Alta):

Cadastro de Novo Usuário:

Tarefa: Preencher formulário de cadastro (nome, email, senha, condomínio).

Tarefa: Validar dados e submeter.

Tarefa: Receber status “Pendente Aprovação” (Morador).

Login/Logout:

Tarefa: Informar email e senha.

Tarefa: Acessar sistema (se aprovado).

Tarefa: Encerrar sessão.

Aprovação de Cadastro (Síndico/Admin):

Tarefa: Visualizar usuários pendentes (do seu condomínio para Síndico, todos para Admin).

Tarefa: Aprovar cadastro.

Tarefa: Rejeitar cadastro.

**Épico 2: Gestão de Anúncios**

Atividades/Features (Release 1 - Prioridade Alta):

Criar Anúncio:

Tarefa: Informar título, descrição, tipo (item/serviço).

Tarefa: Publicar anúncio (fica ativo).

Visualizar Anúncios:

Tarefa: Listar anúncios ativos (filtrados por condomínio).

Tarefa: Ver detalhes de um anúncio específico.

Gerenciar Meus Anúncios:

Tarefa: Listar meus anúncios (ativos e inativos).

Tarefa: Editar anúncio.

Tarefa: Excluir anúncio.

**Épico 3: Gestão de Ofertas**

Atividades/Features (Release 1 - Prioridade Alta):

Fazer Oferta:

Tarefa: Selecionar anúncio.

Tarefa: Escolher tipo de oferta (dinheiro, item, serviço).

Tarefa: Informar valor ou descrição da oferta.

Tarefa: Enviar oferta (status pendente).

Gerenciar Ofertas Recebidas (Anunciante):

Tarefa: Visualizar ofertas pendentes em meus anúncios.

Tarefa: Aceitar oferta (anúncio fica inativo, outras ofertas são recusadas).

Tarefa: Recusar oferta.

Visualizar Minhas Ofertas Enviadas:

Tarefa: Listar ofertas que fiz.

Tarefa: Verificar status de minhas ofertas.

**Épico 4: Administração do Sistema (Admin)**

Atividades/Features (Release 1 - Prioridade Alta):

Gerenciar Usuários:

Tarefa: Listar todos os usuários.

Tarefa: Editar dados de qualquer usuário (incluindo tipo, status, condomínio).

Tarefa: Excluir usuário.

Gerenciar Condomínios:

Tarefa: Criar novo condomínio.

Tarefa: Listar todos os condomínios.

Tarefa: Editar dados de condomínio.

Tarefa: Excluir condomínio.

Atividades/Features (Release 2 - Prioridade Média):

Gerenciar Todos os Anúncios:

Tarefa: Listar todos os anúncios da plataforma.

Tarefa: Editar qualquer anúncio.

Tarefa: Excluir qualquer anúncio.

Visualizar Todas as Ofertas:

Tarefa: Listar/filtrar todas as ofertas da plataforma.

*(Priorização seria indicada visualmente no Story Map, por exemplo, com itens mais acima ou com cores diferentes representando releases/MVPs).*

## 7. ANÁLISE DE PRIORIDADE DE RECURSOS (MODELO KANO SIMPLIFICADO)

Selecionamos três funcionalidades-chave para uma análise simplificada do Modelo Kano:

1. **Funcionalidade-Chave 1: Cadastro e Login de Usuários (RF-001, RF-002)**
   * **Categoria Kano:** **Must-Be (Obrigatória)**
   * **Justificativa:** Sem a capacidade de usuários se registrarem e autenticarem, o sistema não tem como funcionar. É uma expectativa básica e fundamental. A ausência completa dessas funcionalidades causaria extrema insatisfação.
2. **Funcionalidade-Chave 2: Criação e Visualização de Anúncios (RF-008, RF-009)**
   * **Categoria Kano:** **Performance (Unidimensional)**
   * **Justificativa:** A capacidade de criar e ver anúncios é central para o propósito do ShareCondo. Quanto mais fácil, rápido e eficiente for criar um anúncio, e quanto mais relevantes e bem apresentados forem os anúncios listados, maior será a satisfação do usuário. Uma implementação ruim ou limitada causaria insatisfação, enquanto uma boa implementação aumenta a satisfação proporcionalmente.
3. **Funcionalidade-Chave 3: Sistema de Aprovação de Moradores pelo Síndico (RF-007)**
   * **Categoria Kano:** **Attractive (Atratativa) / Performance (para o Síndico)**
   * **Justificativa:**
     + **Para Moradores:** Pode ser vista como **Attractive** ou até mesmo **Must-Be** dependendo da preocupação com segurança no condomínio. Saber que há um controle de quem entra na plataforma específica do condomínio aumenta a confiança e pode ser um diferencial. Se ausente, pode não causar grande insatisfação inicial, mas sua presença agrega valor significativo.
     + **Para Síndicos:** A facilidade e clareza dessa ferramenta é **Performance**. Se for difícil de usar ou confusa, causará insatisfação ao síndico. Se for eficiente, aumenta a satisfação dele com a plataforma.
     + **Para o Negócio/Plataforma:** É um **Must-Be** para garantir a integridade e a proposta de valor de ser uma plataforma “condominial”.

## 8. CASOS DE TESTE FUNCIONAIS

## CT-001: Cadastro de Novo Morador com Sucesso

## ID Requisito Associado: RF-001, RF-006

## Descrição: Verificar se um novo usuário do tipo MORADOR consegue se cadastrar com sucesso selecionando um condomínio.

## Entrada:

## Nome: “João Teste Silva”

## Email: (email único, ex: “joao.teste.novo@exemplo.com”)

## Senha: “senhaValida123”

## Confirmar Senha: “senhaValida123”

## Condomínio: Selecionar um condomínio válido da lista (ex: “Residencial Vila das Flores”).

## Procedimento:

## 1. Acessar a página de cadastro.

## 2. Preencher todos os campos do formulário com os dados de entrada.

## 3. Clicar no botão “Cadastrar”.

## Resultado Esperado:

## 1. O sistema exibe uma mensagem de sucesso: “Cadastro realizado com sucesso! Sua conta passará por aprovação do síndico do condomínio. Você será notificado.”

## 2. O usuário é registrado no banco de dados com o tipo “USUARIO” e status “PENDENTE\_APROVACAO”, associado ao condomínio selecionado.

## Critério de Aceitação: Mensagem de sucesso exibida; usuário consta no banco com dados corretos e status pendente.

## CT-002: Tentativa de Cadastro com Email Duplicado

## ID Requisito Associado: RF-001

## Descrição: Verificar se o sistema impede o cadastro de um novo usuário com um email que já existe.

## Pré-condição: Usuário com email “admin.geral@sharecondo.com” já existe.

## Entrada:

## Nome: “Maria Duplicada”

## Email: “admin.geral@sharecondo.com”

## Senha: “outrasenha456”

## Confirmar Senha: “outrasenha456”

## Condomínio: “Residencial Vila das Flores”

## Procedimento:

## 1. Acessar a página de cadastro.

## 2. Preencher o formulário com os dados de entrada.

## 3. Clicar no botão “Cadastrar”.

## Resultado Esperado:

## 1. O sistema exibe uma mensagem de erro: “Usuário já existe com este e-mail.”

## 2. O usuário não é cadastrado.

## Critério de Aceitação: Mensagem de erro específica sobre email duplicado exibida; nenhum novo usuário é criado.

## CT-003: Login de Usuário Aprovado com Sucesso

## ID Requisito Associado: RF-002

## Descrição: Verificar se um usuário com cadastro APROVADO consegue realizar login com sucesso.

## Pré-condição: Usuário “sindico.flores@exemplo.com” existe, está APROVADO e tem a senha “123”.

## Entrada:

## Email: “sindico.flores@exemplo.com”

## Senha: “123”

## Procedimento:

## 1. Acessar a página de login.

## 2. Preencher email e senha.

## 3. Clicar no botão “Entrar”.

## Resultado Esperado:

## 1. Login é bem-sucedido.

## 2. Usuário é redirecionado para a página “/sindico/aprovar-usuarios”.

## 3. Token JWT e dados do usuário (incluindo tipo SINDICO e nome do condomínio) são armazenados no cliente (localStorage).

## Critério de Aceitação: Redirecionamento correto; token e dados do usuário no localStorage.

## CT-004: Criação de Anúncio de Item por Usuário Aprovado

## ID Requisito Associado: RF-008

## Descrição: Verificar se um usuário MORADOR logado e com status APROVADO consegue criar um novo anúncio do tipo ITEM.

## Pré-condição: Usuário “ana.vflores@exemplo.com” está logado, com status APROVADO, e tem menos de 5 anúncios ativos.

## Entrada (Modal de Criação de Anúncio):

## Título: “Monitor Gamer Usado”

## Descrição: “Monitor 27 polegadas, 144Hz, em perfeito estado.”

## Tipo: “ITEM”

## Procedimento:

## 1. Navegar para a página de Anúncios.

## 2. Clicar em “+ Adicionar Novo Anúncio”.

## 3. Preencher os dados do anúncio no modal.

## 4. Clicar em “Adicionar Anúncio”.

## Resultado Esperado:

## 1. Mensagem de sucesso “Anúncio adicionado com sucesso!” é exibida.

## 2. O novo anúncio aparece na lista de anúncios do usuário (“Meus Anúncios”) e na lista geral de anúncios do seu condomínio.

## 3. O anúncio é persistido no banco de dados associado ao usuário “ana.vflores@exemplo.com”, com status ATIVO e tipo ITEM.

## Critério de Aceitação: Anúncio visível nas listagens corretas; dados corretos no banco.

## CT-005: Síndico Aprova Cadastro de Morador Pendente

## ID Requisito Associado: RF-007

## Pré-condição:

## Usuário “joao.teste.novo@exemplo.com” (do CT-001) está cadastrado com status “PENDENTE\_APROVACAO” e associado ao condomínio “Residencial Vila das Flores”.

## Usuário “sindico.flores@exemplo.com” (Síndico do “Residencial Vila das Flores”) está logado.

## Procedimento:

## 1. Síndico acessa a página “Aprovar Usuários”.

## 2. Localiza o cadastro pendente de “joao.teste.novo@exemplo.com” na lista.

## 3. Clica no botão “Aprovar” para este usuário.

## Resultado Esperado:

## 1. Mensagem de sucesso “Usuário aprovado com sucesso!” é exibida.

## 2. O status do usuário “joao.teste.novo@exemplo.com” é alterado para “APROVADO” no banco de dados.

## 3. O usuário “joao.teste.novo@exemplo.com” desaparece da lista de pendentes.

## 4. O usuário “joao.teste.novo@exemplo.com” agora consegue realizar login com sucesso.

## Critério de Aceitação: Status do usuário atualizado no banco; usuário consegue logar.

## CT-006: Usuário Faz Oferta em Anúncio de Outro Usuário no Mesmo Condomínio

## ID Requisito Associado: RF-012

## Pré-condição:

## Usuário “bruno.vflores@exemplo.com” (Ofertante) está logado e APROVADO, pertence ao “Residencial Vila das Flores”.

## Anúncio “Monitor Gamer Usado” (do CT-004, criado por “ana.vflores@exemplo.com” do mesmo condomínio) está ATIVO.

## Entrada (Modal de Fazer Oferta):

## Tipo de Oferta: “DINHEIRO”

## Valor: 15

## 0.00

## Procedimento:

## 1. Usuário “bruno.vflores@exemplo.com” navega para a página de Anúncios.

## 2. Localiza o anúncio “Monitor Gamer Usado”.

## 3. Clica em “Fazer Oferta”.

## 4. Preenche os dados da oferta no modal.

## 5. Clica em “Enviar Oferta”.

## Resultado Esperado:

## 1. Mensagem de sucesso “Oferta enviada com sucesso!” é exibida.

## 2. A oferta é registrada no banco de dados com status “PENDENTE”, associada ao anúncio e ao ofertante “bruno.vflores@exemplo.com”.

## 3. O anunciante (“ana.vflores@exemplo.com”) pode ver esta oferta em “Meus Anúncios”.

## Critério de Aceitação: Oferta registrada no banco; visível para o anunciante.

## 9. MATRIZES DE RASTREABILIDADE

### 9.1. Matriz de Rastreabilidade: Requisitos vs. Objetivos

Objetivos do Sistema (Exemplos, necessitam ser formalmente definidos para o projeto):

**OBJ-01:** Facilitar a troca e compartilhamento de itens e serviços entre vizinhos de um mesmo condomínio.

**OBJ-02:** Fortalecer o senso de comunidade e colaboração dentro dos condomínios.

**OBJ-03:** Proporcionar uma plataforma segura e controlada para interações condominiais.

**OBJ-04:** Permitir administração centralizada de usuários e condomínios (para ADMINs).

*OBJ-05: Dar autonomia aos Síndicos para gerenciar os moradores de seus condomínios.*

| ID Requisito | OBJ-01 | OBJ-02 | OBJ-03 | OBJ-04 | OBJ-05 |
| --- | --- | --- | --- | --- | --- |
| RF-001 |  | X | X |  |  |
| RF-002 | X | X | X | X | X |
| RF-003 |  |  | X |  |  |
| RF-004 |  |  | X |  |  |
| RF-005 |  |  | X | X |  |
| RF-006 | X |  | X | X |  |
| RF-007 |  | X | X | X | X |
| RF-008 | X | X |  |  |  |
| RF-009 | X | X |  |  |  |
| RF-010 | X |  |  |  |  |
| RF-011 | X |  |  |  |  |
| RF-012 | X | X |  |  |  |
| RF-013 | X |  |  |  |  |
| RF-014 | X |  |  |  |  |
| RF-015 | X |  |  |  |  |
| RF-016 |  |  | X | X |  |
| RNF-001 | X | X |  |  |  |
| RNF-002 | X |  | X |  |  |
| RNF-003 |  |  | X | X | X |
| RNF-004 |  |  | X |  |  |
| RNF-005 | X | X | X | X | X |
| RNF-006 |  |  |  | X | X |
| RNF-007 | X |  |  |  |  |
| RNF-008 | X |  | X | X |  |
| RI-001 | X | X | X | X | X |
| RI-002 | X | X | X | X | X |
| RD-001 |  |  | X | X | X |
| RD-002 |  |  | X | X |  |
| RD-003 | X |  |  |  |  |
| RD-004 | X |  |  |  |  |

### 9.2. Matriz de Rastreabilidade: Requisitos vs. Casos de Teste

| ID Requisito | CT-001 | CT-002 | CT-003 | CT-004 | CT-005 | CT-006 |
| --- | --- | --- | --- | --- | --- | --- |
| RF-001 | X | X |  |  |  |  |
| RF-002 |  |  | X |  |  |  |
| RF-006 | X |  |  |  |  |  |
| RF-007 |  |  |  |  | X |  |
| RF-008 |  |  |  | X |  |  |
| RF-012 |  |  |  |  |  | X |

## 10. GLOSSÁRIO

| Termo | Definição |
| --- | --- |
| **API** | Interface de Programação de Aplicações. Conjunto de rotinas e padrões para acesso a um software ou plataforma. |
| **Back-end** | Parte do sistema responsável pela lógica de negócio, acesso a dados e APIs. No ShareCondo, é Java/Spring Boot. |
| **BCrypt** | Algoritmo de hashing de senhas utilizado para armazenamento seguro. |
| **CORS** | Cross-Origin Resource Sharing. Mecanismo que permite que recursos restritos em uma página web sejam recuperados por outro domínio. |
| **CRUD** | Acrônimo para as quatro operações básicas de persistência de dados: Create (Criar), Read (Ler), Update (Atualizar), Delete (Excluir). |
| **DTO** | Data Transfer Object. Objeto simples usado para transferir dados entre camadas de um sistema, como entre o controller e o cliente. |
| **Endpoint** | URL específica onde uma API pode ser acessada. |
| **Front-end** | Parte do sistema com a qual o usuário interage diretamente. No ShareCondo, é React. |
| **JWT** | JSON Web Token. Padrão aberto para criar tokens de acesso que afirmam um número de declarações. Usado para autenticação. |
| **Maven** | Ferramenta de automação de compilação e gerenciamento de dependências para projetos Java. |
| **Modal** | Janela de diálogo que aparece sobre o conteúdo principal da página para solicitar informações ou ações. |
| **React** | Biblioteca JavaScript para criar interfaces de usuário. |
| **RESTful** | Estilo arquitetural para criação de APIs web, utilizando os verbos HTTP (GET, POST, PUT, DELETE). |
| **Spring Boot** | Framework Java que facilita a criação de aplicações Spring autônomas e prontas para produção. |
| **SQL Server** | Sistema de gerenciamento de banco de dados relacional da Microsoft. |
| **Token** | Sequência de caracteres usada para autenticar e autorizar um usuário após o login. |
| **UI** | User Interface. Meio pelo qual o usuário interage com o sistema. |
| **UX** | User Experience. Experiência geral de uma pessoa ao usar um produto, especialmente em termos de quão fácil ou agradável é de usar. |

## 11. REFERÊNCIAS BIBLIOGRÁFICAS E FERRAMENTAS

**Tecnologias e Frameworks:**

Back-end:

* Java 17
* Spring Boot 3.4.5
* Spring Security
* Spring Data JPA
* Hibernate (via Spring Data JPA)
* java-jwt (Auth0) 4.5.0
* Maven (para gerenciamento de dependências e build)

Front-end:

* React 19.1.0
* React Router DOM 7.6.0
* Axios 1.9.0 (para requisições HTTP)
* Create React App (como base do projeto front-end)

Banco de Dados:

* Microsoft SQL Server

**Ferramentas de Desenvolvimento:**

* IDE Java: IntelliJ IDEA
* IDE JavaScript/React: VS Code
* Node.js e npm (para o ambiente de desenvolvimento front-end)
* Git (para controle de versão)
* Postman (para testes de API)
* Navegadores Web (Chrome, Firefox, etc., para desenvolvimento e teste do front-end)

**Outras Referências:**

* Princípios de Design de API REST.
* Padrões de Projeto de Software.
* Documentação oficial das tecnologias listadas.
* MODELO DE DOCUMENTAÇÃO DE REQUISITOS.docx (fornecido como base para este documento). [cite: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]
* popula\_banco.py: Script Python para popular o banco de dados com dados de teste.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Apêndices (Conforme modelo, seriam incluídos aqui):**

**Apêndice A: Modelo de Dados (Diagrama Entidade-Relacionamento) - [ESPAÇO RESERVADO - INSERIR DIAGRAMA AQUI] [cite: 31]**

**Apêndice B: Protótipos de Interface (Mais detalhados, se necessário além do código) - [ESPAÇO RESERVADO - INSERIR WIREFRAMES/MOCKUPS AQUI] [cite: 31]**

**Apêndice C: Casos de Uso (Diagramas e descrições detalhadas) - [ESPAÇO RESERVADO - INSERIR DIAGRAMAS E DETALHAMENTO DOS CASOS DE USO AQUI] [cite: 31]**