Plano de Testes: Aplicação Reside (Backend API)

1. Introdução

Reside é um sistema de gerenciamento de condomínios, com o propósito de centralizar e facilitar a vida em comunidade, oferecendo uma plataforma robusta para a gestão de recursos e comunicação.

Este documento detalha o plano de testes para garantir a qualidade, estabilidade e corretude da lógica de negócio da API do Reside. As principais funcionalidades cobertas incluem:

- **Gestão de Moradores e Autenticação:** Cadastro, login e recuperação de informações de perfil.
- **Reservas de Áreas Comuns (Bookings):** Agendamento e validação de regras de uso.
- **Mensageria Interna:** Comunicação direta entre moradores e avisos gerais para o condomínio.
- **Gestão de Vagas de Garagem (Parking):** Disponibilização e solicitação de vagas.
- **Solicitações e Ocorrências (Requests):** Abertura e acompanhamento de chamados.

2. Arquitetura

A API do Reside é construída sobre uma stack moderna de Node.js, seguindo os princípios da Arquitetura Limpa (Clean Architecture) para garantir o desacoplamento e a testabilidade das camadas.

- **Framework/Runtime:** Node.js com Express.js para a criação de endpoints RESTful.
- **Linguagem:** JavaScript (ES Modules).
- **Camada de Domínio (Core):** Contém toda a lógica de negócio pura, encapsulada em Serviços de Aplicação (ex: CreateBookingService). É o foco principal dos testes unitários.
- **Camada de Infraestrutura:** Responsável pela comunicação com o mundo externo.
    - **Banco de Dados:** Utiliza o ORM **Sequelize** para interagir com um banco de dados relacional PostgreSQL. As implementações concretas dos Repositórios residem aqui.
    - **Mensageria Assíncrona:** Utiliza **Kafka (Redpanda)** para publicação de eventos críticos (ex: novas mensagens), garantindo a resiliência e o desacoplamento de serviços futuros (como notificações push).

- **Camada de Apresentação:** Controladores (Controllers) que recebem requisições HTTP, validam os dados de entrada (payload, params) e orquestram as chamadas aos Serviços de Aplicação.

3. Funcionalidades e Critérios de Aceite

A seguir, detalhamos as funcionalidades da API e seus respectivos critérios de aceite, que guiarão a criação dos casos de teste.

Funcionalidade	**Criação de Reserva (Booking)**
Comportamento Esperado	Um morador autenticado deve poder criar uma reserva para uma área comum enviando uma requisição POST para o endpoint /bookings. Requisições bem-sucedidas retornam a reserva criada (201 Created); falhas retornam códigos de erro específicos.
Critérios de Aceite	• Retornar 400 Bad Request se a data de término (endTime) for anterior ou igual à data de início (startTime). • Retornar 400 Bad Request se a data da reserva for no passado. • Retornar 400 Bad Request se a reserva estiver fora do horário permitido (10h-22h). • Retornar 409 Conflict se já existir uma reserva que se sobreponha ao horário solicitado para a mesma área. • Retornar 201 Created com os dados da reserva em caso de sucesso. • O amenityId e o residentId devem ser validados.

Funcionalidade	**Gestão de Moradores (Residents)**
Comportamento Esperado	O sistema deve gerenciar as informações dos moradores, incluindo um fluxo de "verificar ou criar" no primeiro login via Google e a capacidade de buscar informações formatadas do perfil.
Critérios de Aceite	• No fluxo checkOrCreateUser, se o google_id existir, retornar o morador. Se não, criar um novo morador com registered: false. • getUserInfoByGoogleId deve retornar um objeto formatado (DTO) com dados essenciais do morador e o nome do seu condomínio. • getUserInfoByGoogleId deve retornar 404 Not Found se o morador não for encontrado. • Deve ser possível atualizar os dados de um morador.

Funcionalidade	**Envio de Mensagens (Messaging)**

Comportamento Esperado	Moradores devem poder enviar mensagens diretas para outros moradores ou enviar um aviso geral para todos no condomínio. O sistema deve suportar mensagens de texto e imagens.
Critérios de Aceite	• Criar um registro no banco de dados para cada mensagem direta. • Publicar um evento no tópico Kafka RESIDENT_MESSAGES para cada mensagem direta. • Para avisos gerais, criar um registro de mensagem para **cada morador** do condomínio. • Para avisos gerais, publicar um **único** evento no tópico Kafka CONDOMINIUM_MESSAGES. • Imagens devem ser comprimidas antes de serem salvas. • Uma falha na publicação para o Kafka não deve impedir o salvamento da mensagem no banco de dados (deve apenas logar um aviso).

Funcionalidade	**Gestão de Vagas de Garagem (Parking)**
Comportamento Esperado	Moradores podem anunciar suas vagas de garagem para aluguel (diário ou mensal) e outros moradores podem solicitar essas vagas. O sistema deve validar regras de negócio e gerenciar o ciclo de vida das vagas.
Critérios de Aceite	• Um apartamento não pode ter mais de 2 vagas anunciadas simultaneamente. • A criação de uma vaga deve validar todos os campos obrigatórios (localização, tipo, preço, etc.). • Vagas do tipo diario não podem ser criadas para datas passadas. • Vagas do tipo mensal devem ter ao menos um dia da semana selecionado. • Um morador pode solicitar uma vaga com status disponivel. • Um processo automatizado (checkExpiredReservations) deve ser capaz de cancelar reservas expiradas.

4. Estratégia de Teste

- **Escopo de Testes**
  O plano de testes abrange todas as funcionalidades da API do Reside descritas na seção anterior. O foco é garantir a robustez da lógica de negócio e a estabilidade das integrações.
- **Níveis de Teste**
  - **Testes Unitários:** Foco principal na **camada de Domínio/Core (src/core/services)**. O objetivo é ter uma cobertura de testes superior a **90%** para a lógica de negócio. Todas as dependências externas (repositórios, Kafka, Firebase, etc.) serão substituídas por **Mocks** para garantir testes rápidos e isolados.

        - *Responsabilidade:* Time de Desenvolvimento.

    - **Testes Manuais:** Testes exploratórios realizados para encontrar bugs em cenários complexos ou não cobertos pela automação.
        - *Responsabilidade:* Time de QA.

5. Ambiente e Ferramentas

- **Ambiente de Testes:** Os testes serão executados em um ambiente de homologação, que é uma réplica do ambiente de produção. As dependências (PostgreSQL, Redpanda/Kafka) serão orquestradas via **Docker Compose** para consistência.
- **Ferramentas:**

Ferramenta	Time	Descrição
Jest	Desenvolvimento	Framework principal para a execução de **Testes Unitários**.
Postman	QA / Desenvolvimento	Ferramentas para **Testes Manuais e Exploratórios** da API.

6. Classificação de Bugs

Os bugs serão classificados de acordo com a seguinte escala de severidade:

ID	Nível de Severidade	Descrição
1	**Blocker**	• Um bug que impede a execução de testes ou o uso de uma funcionalidade crítica. • Um endpoint principal retorna erro 500 ou está inacessível. • Corrupção de dados. • **Bloqueia a entrega (release).**
2	**Grave**	• Uma funcionalidade principal não se comporta como o esperado. • Um endpoint retorna dados incorretos ou não executa a lógica de negócio corretamente. • Falhas de validação que permitem a entrada de dados inconsistentes.
3	**Moderada**	• Uma funcionalidade secundária não se comporta como o esperado. • Mensagens de erro da API não são claras ou não seguem o padrão.

		• O desempenho de um endpoint está abaixo do aceitável, mas não bloqueia o uso.
4	**Pequena (Minor)**	• Erros de digitação em mensagens de retorno. • Pequenas inconsistências na estrutura do JSON de resposta que não quebram o contrato com o cliente. • Otimizações ou melhorias de baixo impacto.

7. Definição de Pronto (Definition of Done)

Uma funcionalidade (ou estória de usuário) será considerada **"Pronta"** para entrega quando todos os seguintes critérios forem atendidos:

- O código foi implementado e revisado (Code Review) por outro membro da equipe.
- Todos os **Testes Unitários** estão passando.
- A **cobertura de testes** para a lógica de negócio nova ou alterada está acima de **90%**.
- Nenhum bug com severidade Blocker ou Grave associado à funcionalidade está aberto.
- A documentação da API (e.g., Swagger/OpenAPI) foi criada ou atualizada.
- A funcionalidade foi validada em ambiente de homologação.