Test Plan Chama o Síndico

1 Introdução

Chama o Síndico é um sistema que visa facilitar a organização do condomínio, principalmente para o síndico. O sistema consta com funcionalidades como, chat em tempo real, registro de encomenda, manutenção, reservas de áreas comuns, controle de entrada e saída de visitante e integração com sistema de notificações para manter a boa visibilidade para os usuários.

2 Arquitetura

O Flutter é utilizado para desenvolver a interface de usuário das aplicações, tanto para dispositivos móveis quanto para a Web. Essas aplicações se comunicam com uma API desenvolvida em Node.js, que, por sua vez, está integrada a um banco de dados PostgreSQL. A aplicação mobile também está integrada ao Firebase Cloud Messaging (FCM), utilizado para o envio de notificações push. Além disso, o Node.js se comunica com o Apache Kafka, que é utilizado como mecanismo de mensageria assíncrona e processamento de eventos, possibilitando a construção de fluxos de dados distribuídos e desacoplados dentro do sistema.

3 Funcionalidades

Funcionalidade	Registro Ocorrência
Comportamento Esperado	● O serviço deve criar um novo registro de ocorrência usando os dados fornecidos (descrição, título, dataCriacao). Deve

	adicionar o userId ao registro. Deve salvar o registro no banco de dados via repositório. Deve retornar o registro criado com ID gerado. Em caso de erro durante o salvamento, deve lançar erro.
Verificações	● Verifica se o resultado retornado é igual ao mock esperado ● Verifica se repo.create foi chamado com os parâmetros corretos ({ ...dto, userId }) ● Verifica se repo.save foi chamado com o registro criado ● Verifica se o userId foi adicionado ao registro ● Verifica se BadRequestException é lançada quando repo.save falha ● Verifica se o erro é propagado corretamente.
Critérios de Aceite	● O método create deve aceitar um DTO e userId como parâmetros ● Deve criar uma nova instância de RegistroOcorrencia com os dados fornecidos ● Deve incluir o userId no registro criado ● Deve salvar o registro no banco de dados usando o repositório ● Deve retornar o registro salvo com ID gerado ● Deve capturar erros de salvamento no banco

	● Deve lançar erro em caso de falha ● Não deve retornar dados parciais em caso de erro

Funcionalidade	Gerenciamento de Encomenda
Comportamento Esperado	● O serviço deve criar uma nova encomenda usando os dados fornecidos (descrição, estimatedDelivery). Deve associar o pacote ao apartamento através do apartamentoId. Deve definir o status inicial como PENDENTE. Deve salvar o pacote no banco de dados via repositório. Deve retornar o pacote criado com ID gerado. Deve validar se apartamentoId foi informado e lançar erro se não fornecido.
Verificações	● Verifica se o resultado retornado é igual ao mock esperado ● Verifica se repo.create foi chamado com os parâmetros corretos ({ ...dto, apartamento: { id: apartamentoId }, status: packageStatus.PENDENTE }) ● • Verifica se o apartamentoId foi associado corretamente ao pacote ● Verifica se o status inicial é definido como PENDENTE ● Verifica se ForbiddenException é lançada quando apartamentoId não é informado

	● Verifica se a validação de apartamentoId está funcionando corretamente
Critérios de Aceite	● O método create deve aceitar um DTO e apartamentoId como parâmetros ● Deve criar uma nova instância de Package com os dados fornecidos ● Deve associar o pacote ao apartamento correto através do apartamentoId ● Deve definir automaticamente o status inicial como PENDENTE ● Deve salvar o pacote no banco de dados usando o repositório ● Deve retornar o pacote salvo com ID gerado ● Deve validar a obrigatoriedade do apartamentoId ● Deve lançar erro se apartamentoId for undefined/null ● Não deve permitir criação de pacote sem associação a apartamento

Funcionalidade	Criar Manutencao
Comportamento Esperado	● O serviço deve criar uma nova manutenção usando os dados fornecidos (descrição, dataManutencao,

	dataProximaManutencao, manutencaoRealizada). Deve associar a manutenção ao apartamento por meio do apartamentoId. Deve definir o status inicial como PENDENTE ou CONCLUIDO (se manutencaoRealizada for true). Deve salvar a manutenção no banco de dados via repositório. Deve retornar a manutenção criada com ID gerado. Deve validar se o apartamentoId foi informado e lançar erro se não fornecido.
Verificações	● Verifica se o resultado retornado é igual ao mock esperado ● Verifica se repository.create foi chamado com os parâmetros corretos: { ...dto, apartamento: { id: apartamentoId }, status: manutencaoStatus.PENDENTE } ● Verifica se o apartamentoId foi corretamente associado à manutenção ● Verifica se o status inicial é definido como PENDENTE ou CONCLUIDO conforme manutencaoRealizada ● Verifica se ForbiddenException é lançada quando apartamentoId não é informado ● Verifica se a validação da obrigatoriedade do apartamentoId está funcionando corretamente
Critérios de Aceite	● O método create deve aceitar um DTO e apartamentoId como parâmetros

	● Deve criar uma nova instância de Manutencao com os dados fornecidos ● Deve associar a manutenção ao apartamento correto por meio do apartamentoId ● Deve definir automaticamente o status inicial como PENDENTE (ou CONCLUIDO se manutencaoRealizada = true) ● Deve salvar a manutenção no banco de dados usando o repositório ● Deve retornar a manutenção salva com ID gerado ● Deve validar a obrigatoriedade do apartamentoId ● Deve lançar erro se apartamentoId for undefined ou null ● Não deve permitir criação de manutenção sem associação ao apartamento

Funcionalidade	Remover Reserva
Comportamento Esperado	● O serviço deve permitir remover uma reserva existente. Deve buscar a reserva pelo ID. Se encontrada, deve removê-la via repositório. Se não for encontrada, deve lançar uma exceção (Booking with id X not found).

Verificações	• Verifica se repository.findOne é chamado com o ID correto • Verifica se repository.delete é chamado com o ID da reserva existente • Verifica se remove() resolve com sucesso quando a reserva existe • Verifica se uma exceção é lançada quando a reserva não existe
Critérios de Aceite	• O método remove deve aceitar um ID como parâmetro • Deve buscar a reserva pelo ID fornecido • Deve chamar delete() no repositório se a reserva existir • Deve lançar exceção se a reserva não existir • Não deve permitir remoção de reservas inexistentes

Funcionalidade	Encontrar Áreas Comuns
Comportamento Esperado	• O serviço deve retornar todas as áreas cadastradas no sistema. Caso ocorra algum erro na busca, deve lançar uma exceção com a mensagem apropriada.
Verificações	• Verifica se o método repository.findAll foi chamado corretamente

	• Verifica se o resultado retornado é igual ao mock (lista de áreas) • Verifica se uma exceção é lançada em caso de erro na consulta
Critérios de Aceite	• O método findAll deve retornar um array de objetos Area • Deve chamar o método findAll() do repositório • Deve lançar erro se a busca falhar • Não deve retornar undefined ou resultado inesperado

Funcionalidade	Criar Aviso
Comportamento Esperado	O serviço deve criar um novo aviso usando os dados fornecidos (título, mensagem, data). Apenas usuários autorizados (síndico/funcionário) podem criar avisos. O aviso deve ser salvo no banco de dados via repositório e retornar o objeto criado com ID gerado. Em caso de erro, deve lançar exceção apropriada.
Verificações	• Verifica se o resultado retornado é igual ao mock esperado. • Verifica se `repo.create` foi chamado com os parâmetros corretos. • Verifica se `repo.save` foi chamado com o aviso criado. - Verifica se apenas

	usuários autorizados conseguem criar avisos. ● Verifica se a exceção é lançada em caso de erro.
Critérios de Aceite	● O método `create` deve aceitar um DTO como parâmetro. ● Deve criar uma nova instância de Aviso com os dados fornecidos. ● Deve salvar o aviso no banco de dados usando o repositório. ● Deve retornar o aviso salvo com ID gerado. ● Deve lançar erro em caso de falha. ● Não deve permitir criação por usuários não autorizados.

Funcionalidade	Buscar Apartamento por ID
Comportamento Esperado	O controlador deve retornar os dados do apartamento correspondente ao ID fornecido, caso ele exista. Se o apartamento não for encontrado, deve lançar uma exceção do tipo NotFoundException.
Verificações	● Verifica se o método findOne do serviço foi chamado com o ID correto.

	• Verifica se o valor retornado pelo controlador é igual ao mock esperado quando o ID é válido. • Verifica se NotFoundException é lançado corretamente quando o serviço retorna undefined.
Critérios de Aceite	• O método findOne do controlador deve aceitar um ID como string. • Deve chamar o método findOne do serviço com o ID convertido (se necessário). • Deve retornar o objeto Apartamento quando encontrado. • Deve lançar NotFoundException se nenhum apartamento for encontrado. • Não deve retornar undefined nem valores inválidos em caso de erro.

Funcionalidade	Funcionalidade: Retornar Vagas por Apartamento
Comportamento Esperado	O controlador deve retornar uma lista de vagas associadas ao apartamento cujo ID é informado. Caso não existam vagas registradas para esse apartamento, o sistema deve lançar uma exceção do tipo NotFoundException.
Verificações	• Verifica se o método findByApartamento do serviço é chamado com o ID correto.

	● Verifica se o retorno do método do controlador é equivalente ao mock esperado (array de vagas). ● Verifica se NotFoundException é lançada quando o serviço retorna undefined.
Critérios de Aceite	● O método findByApartamento deve aceitar um apartamentoId como parâmetro. ● Deve chamar o serviço findByApartamento com esse ID. ● Deve retornar um array de vagas associadas ao apartamento. ● Deve lançar NotFoundException se nenhuma vaga for encontrada. ● Não deve retornar valores indefinidos ou inesperados. ● Deve tratar corretamente o caso em que não há vagas vinculadas ao apartamento informado.

Funcionalidade	Funcionalidade: Buscar Visitantes por Apartamento
Comportamento Esperado	O controlador deve retornar todos os visitantes associados ao apartamento cujo apartamentoId for informado. Caso não existam visitantes para o

	apartamento ou o ID seja inválido, o sistema deve lançar uma exceção HTTP com código 404.
Verificações	• Verifica se o método findVisitantesByApartamento do serviço foi chamado com o ID correto. • Verifica se o valor retornado é equivalente ao mock esperado (lista de visitantes). • Verifica se HttpException com status 404 - Not Found é lançada corretamente quando não há visitantes encontrados. • Verifica se o apartamentoId é validado pelo serviço de apartamento (ApartamentoService.findOne).
Critérios de Aceite	• O método findVisitantesByApartamento deve aceitar um apartamentoId como string. • Deve consultar o ApartamentoService.findOne para validar se o apartamento existe. • Deve chamar o método findVisitantesByApartamento do serviço com o apartamentoId. • Deve retornar a lista de visitantes vinculados ao apartamento. • Deve lançar HttpException com status 404 caso nenhum visitante seja encontrado.

	● Não deve retornar undefined ou dados incompletos. ● Deve garantir que o apartamento existe antes de buscar os visitantes.

4 Estratégia de Teste

- **Escopo de Testes**

O plano de testes abrange todas as funcionalidades descritas na tabela acima.

Serão executados testes em todos os níveis conforme a descrição abaixo.

Testes Unitários: o código terá uma cobertura de 60% de testes unitários, que são de responsabilidade dos desenvolvedores.

- **Ambiente e Ferramentas**

Os testes serão feitos do ambiente de homologação, e contém as mesmas configurações do ambiente de produção com uma massa de dados gerada previamente pelo time de qualidade.

As seguintes ferramentas serão utilizadas no teste:

Ferramenta	**Time**	**Descrição**
Insomnia	Qualidade	Ferramenta para realização de testes de API
Jest	Desenvolvimento	Framework utilizada para testes unitários

5 Classificação de Bugs

Os Bugs serão classificados com as seguintes severidades:

ID	Nivel de Severidade	Descrição
1	Blocker	● Bug que bloqueia o teste de uma função ou feature causa crash na aplicação. ● Botão não funciona impedindo o uso completo da funcionalidade. ● Bloqueia a entrega.
2	Grave	● Funcionalidade não funciona como o esperado ● Input incomum causa efeitos irreversíveis
3	Moderada	● Funcionalidade não atinge certos critérios de aceitação, mas sua funcionalidade em geral não é afetada ● Mensagem de erro ou sucesso não é exibida
4	Pequena	● Quase nenhum impacto na funcionalidade porém atrapalha a experiência ● Erro ortográfico ● Pequenos erros de UI

6 Definição de Pronto

Será considerada pronta as funcionalidades que passarem pelas verificações e testes descritas nestes TestPlan, não apresentarem bugs com a severidade acima de Minor, e passarem por uma validação de negócio de responsabilidade do time de produto.