Solution Design Wiki: TurboCat CatCar

1. INTRODUÇÃO

Este documento, intitulado "Solution Design Wiki: TurboCat Core", tem como objetivo principal detalhar a concepção e o design de uma solução de software para a oficina TurboCat. Ele serve como um guia abrangente e um material de consulta contínua para a equipe de desenvolvimento e arquitetura, garantindo alinhamento e consistência durante todas as fases do projeto.

Apresenta os princípios de design estratégico que guiaram as decisões iniciais, o escopo do Produto Mínimo Viável (MVP), a análise dos conceitos de negócio e hipóteses, a definição do Core Domain, o mapeamento do Event Storming e as relações entre os Bounded Contexts.

2. DESIGN ESTRATÉGICO

2.1 Perfil da Equipe

- **Composição:** A equipe é composta por 1 Arquiteto, 1 Desenvolvedor Sênior e a IA Orquestradora.
- **Características:** Trata-se de um time extremamente enxuto, de alta senioridade e com foco em automação. A comunicação é direta e de baixo custo.

2.2 Decisão Conway/Inverse Conway

- **Análise:** Com uma única unidade de desenvolvimento, a arquitetura pode ser coesa. No entanto, existe a visão de uma futura evolução para um sistema distribuído.
- **Decisão:** Adotaremos uma abordagem de **Monólito Modular com Máximo Desacoplamento**. A arquitetura será unificada para acelerar a entrega inicial, mas os módulos serão projetados com fronteiras claras e comunicação via interfaces bem definidas (Portas), preparando o terreno para uma eventual extração para microsserviços.

2.3 Escopo Imediato (MVP em 1 Mês)

- **Foco Principal:** O escopo deste MVP é a entrega de uma aplicação exclusivamente backend, expondo todas as suas funcionalidades através de uma API RESTful bem definida. Não haverá desenvolvimento de interface de usuário (frontend).
    - **A. Criação da Ordem de Serviço (OS):**
        - Endpoints para identificação/criação de cliente por CPF/CNPJ.

        - Endpoints para cadastro de veículo.
        - Endpoints para criação de OS, incluindo serviços, peças e insumos.
        - Lógica para geração automática de orçamento com base nos itens da OS.
        - Endpoint para registrar a aprovação do orçamento pelo cliente.
    -
    - **B. Acompanhamento da OS:**
        - Gestão do ciclo de vida com os status: Recebida, Em diagnóstico, Aguardando aprovação, Em execução, Finalizada, Entregue.
        - Lógica para alteração automática dos status.
        - Endpoint de consulta pública (para clientes) para acompanhar o progresso da OS.
    -
    - **C. Gestão Administrativa:**
        - API de CRUD para clientes, veículos, serviços e peças (com controle de estoque).
        - Endpoints para listagem e detalhamento de ordens de serviço.
        - Endpoint para monitoramento de dados de tempo médio de execução.
    -
    - **D. Segurança e Qualidade:**
        - Implementação de autenticação JWT para proteger as APIs administrativas.
        - Validação de dados sensíveis em todas as camadas.
        - Implementação de testes unitários e de integração para os endpoints.
    -

2.4 Princípios de Design

- **Design API-First:** A API não é um subproduto; ela é o produto deste MVP. Deve ser projetada com clareza, consistência e boas práticas RESTful.
- **Priorização Agressiva Dentro do Backend:** A equipe deve priorizar o fluxo principal da OS (Criação e Acompanhamento), tratando as funcionalidades de gestão como secundárias.
- **Design para Desacoplamento:** Cada módulo (ex: "Gestão de OS", "Estoque", "Clientes") deve ser o mais independente possível, com comunicação baseada em contratos e eventos.
- **Segurança e Testes desde o Início:** Requisitos de JWT, validação e cobertura de testes são partes integrantes da entrega de cada endpoint.
- **DDD Focado no Essencial:** Aplicar DDD com rigor no Core Domain. A Linguagem Ubíqua é inegociável.
- **Automação Intensiva:** A IA será usada agressivamente para gerar boilerplate de APIs, testes, e acelerar o desenvolvimento.

2.5 Declaração de Idioma do Projeto

- **Idioma Selecionado:** Inglês (en-US).
- **Regra:** Todos os artefatos de código e de domínio devem, obrigatoriamente, ser escritos em inglês.

2.6 Lista de Conceitos de Negócio e Hipóteses

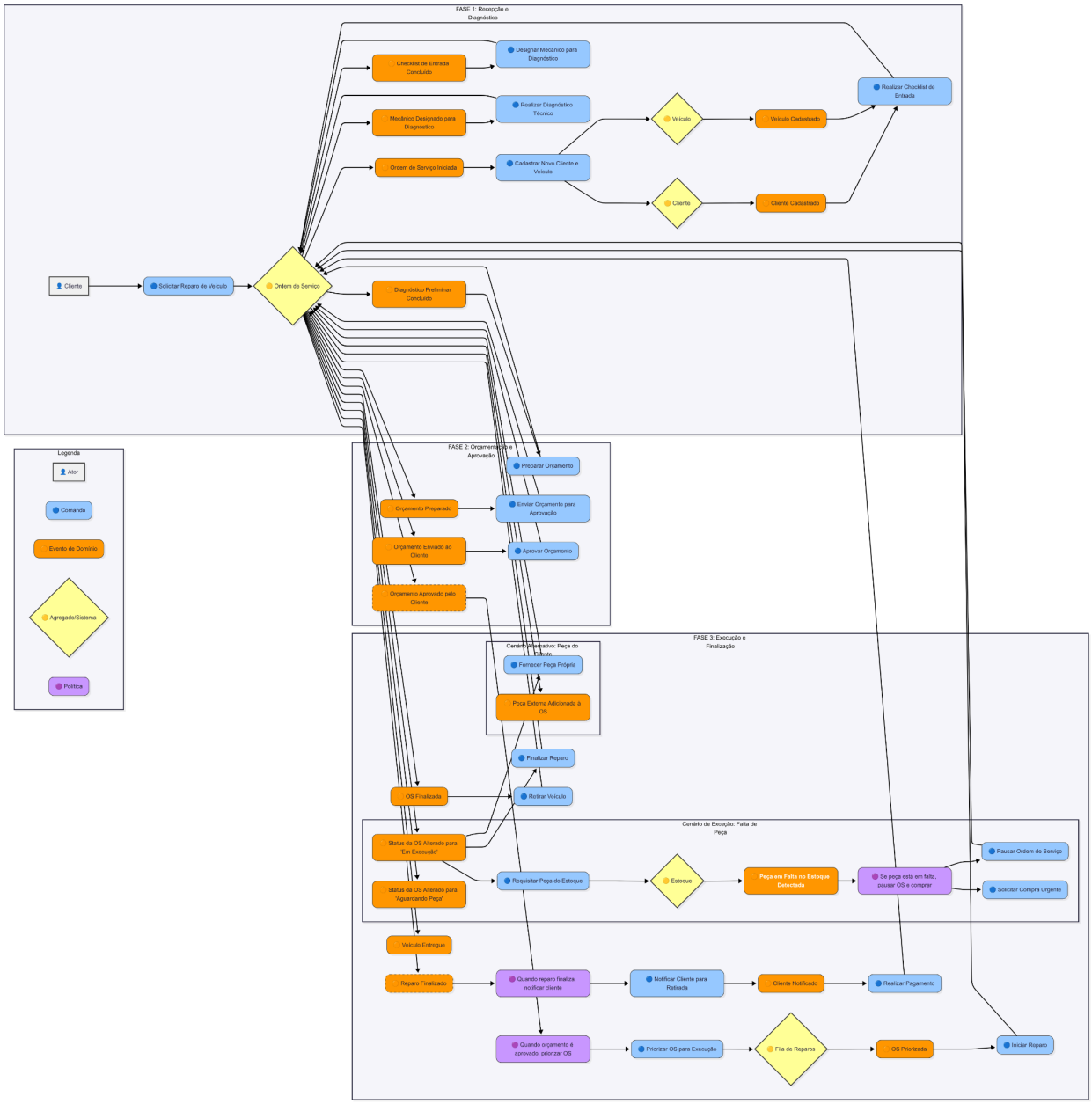
- **Atores e Papéis Principais (Actors & Roles):** Cliente, Funcionários Administrativos, Gerentes, Mecânico, Fornecedor, Guincheiro.
- **Artefatos e Documentos Chave (Key Artifacts):** Ordem de Serviço (OS), Checklist de Entrada, Checklist Técnico, Orçamento, Veículo, Peça, Estoque, Garantia.
- **Processos e Verbos do Domínio (Domain Processes & Verbs):** Recepção, Cadastro, Diagnóstico, Designar mecânico, Preparar Orçamento, Aprovação do Cliente, Execução do Reparo, Priorização, Consultar Status.
- **Conceitos e Linguagem Ubíqua (Ubiquitous Language & Concepts):** Serviço de Reparo de Confiança, Transparência, Ordem de Chegada, Peça Fornecida pelo Cliente, Status da OS.
- **Dores e Hipóteses de Melhoria (Pains & Hypotheses):** Perda de histórico, Falha no controle de estoque, Dificuldade em acompanhar status, Priorização reativa, Fluxo de orçamento ineficaz.

2.7 Definição do Core Domain

- **Core Domain: Gerenciamento do Ciclo de Vida do Reparo Confiável (Reliable Repair Lifecycle Management)**
    - **Definição:** O Core Domain é o processo de ponta a ponta que garante que um veículo seja reparado com precisão técnica, qualidade e comunicação transparente, gerenciando e fortalecendo a confiança do cliente. Engloba diagnóstico, orçamento, aprovação, execução e acompanhamento transparente.
- **Justificativa de Negócio:** O verdadeiro ativo da TurboCat é a confiança. O "Serviço de Reparo de Confiança" é o coração do negócio. O software deve ser modelado para proteger e ampliar essa capacidade. Processos como gestão de estoque ou cadastro de clientes são Domínios de Suporte.
- **Diretrizes Estratégicas de Modelagem:** Foco na complexidade essencial, concentrando esforços de modelagem (Aggregates, Events) neste Core Domain.

2.8 Mapa do Event Storming

Diagrama de Fluxo



Legenda do Diagrama

- 👤 **Ator** (Cinza): Pessoa ou sistema que inicia uma ação.
- 🔵 **Comando** (Azul): Ação ou intenção de um usuário que o sistema deve processar.
- 🟠 **Evento de Domínio** (Laranja): Fato relevante para o negócio que ocorreu no passado.
- 🟡 **Agregado/Sistema** (Amarelo): Onde as regras de negócio são aplicadas e os eventos são gerados.
- 🟣 **Política** (Roxo/Lilás): Regra de negócio do tipo "Sempre que [Evento X] acontecer, então [Comando Y]".

2.9 Declaração de Propósito da Aplicação e KPIs

- **Missão da Aplicação:** Empoderar a oficina TurboCat com uma ferramenta digital que transforme seu "caos funcional" em uma eficiência transparente, automatizando tarefas repetitivas e fornecendo dados precisos para fortalecer o "Serviço de Reparo de Confiança" que é o coração do seu negócio.
- **Problemas Prioritários a Resolver (Foco do MVP) e KPIs de Sucesso:**
    - **CRÍTICO - Visibilidade do Serviço:**
        - **Problema:** Eliminar a incerteza sobre o andamento dos reparos.
        - **KPI Associado:** Reduzir em 50% o número de ligações de clientes perguntando sobre o status da OS em até 3 meses.
    -
    - **ALTO - Confiabilidade do Inventário:**
        - **Problema:** Garantir a acurácia do inventário para evitar paralisações.
        - **KPI Associado:** Reduzir o "Tempo de Espera por Peça" em 70% para itens em estoque.
    -
    - **MÉDIO - Eficiência do Fluxo de Trabalho:**
        - **Problema:** Agilizar a criação de valor.
        - **KPI Associado:** Diminuir em 30% o tempo médio entre a "Ordem de Serviço Iniciada" e o "Orçamento Aprovado pelo Cliente".

2.10 Definição das Relações do Context Map

- **Relação #1: Front Office -> Workshop (Pub/Sub):** Front Office (Upstream) publica QuoteApprovedByCustomer. Workshop (Downstream) assina o evento.
- **Relação #2: Workshop e Inventário (OHS/PL):** Inventário (Upstream) expõe uma API. Workshop (Downstream) consome para consultar e requisitar peças.
- **Relação #3: Front Office e Inventário (OHS/PL):** Inventário (Upstream) expõe a mesma API. Front Office (Downstream) consome para obter preços para o orçamento.
- **Relação #4: Workshop -> Front Office (Pub/Sub):** Workshop (Upstream) publica eventos de progresso (RepairExecutionStarted, RepairCompleted). Front Office (Downstream) assina para atualizar o status da OS.

2.11 Matriz de Rastreabilidade (Problemas vs. Componentes)

Problema de Negócio	Componente de Software Responsável
Visibilidade do Serviço	Agregado WorkOrder (status), Agregado RepairJob (status técnico), Eventos de status (Pub/Sub).

Confiabilidade do Inventário	Contexto Inventory & Procurement, Agregado InventoryItem, Agregado PurchaseOrder.
Eficiência do Fluxo de Trabalho	Agregado Customer (histórico), Agregado WorkOrder (fluxo de orçamento), Contratos de Leitura (DTOs).

2.12 Subdomínios Identificados (O Espaço do Problema)

Esta é a decomposição do negócio da TurboCat em suas áreas lógicas, classificadas por sua importância estratégica.

- **Core Domain (Domínio Central):**
  - **Nome:** Gerenciamento do Ciclo de Vida do Reparo Confiável
  - **Descrição:** O processo de ponta a ponta que engloba o diagnóstico técnico, a criação e aprovação do orçamento, a execução do reparo e o acompanhamento transparente do status. É a principal fonte de valor e diferencial competitivo da oficina: a confiança do cliente.
- **Supporting Subdomains (Domínios de Suporte):**
  - **Nome:** Gestão de Inventário e Aquisições
  - **Descrição:** Controla o estoque de peças, gerencia fornecedores e o processo de aquisição de novos itens. É vital para a operação, mas não é o diferencial competitivo. É um problema complexo, mas já resolvido (um "solved problem").
  - **Nome:** Gestão de Clientes e Veículos
  - **Descrição:** Gerencia o cadastro, o histórico e os dados de contato dos clientes, bem como as informações dos seus veículos. É um CRM simplificado, necessário para habilitar o Core Domain.
  - **Nome:** Faturamento e Pagamentos (Billing)
  - **Descrição:** Lida com a geração de faturas (invoices), processamento de pagamentos e registros financeiros. É um processo financeiro padrão, essencial para o negócio, mas não é único.
- **Generic Subdomain (Domínio Genérico):**
  - **Nome:** Comunicações Transacionais
  - **Descrição:** O envio de notificações (e-mail, SMS) para os clientes sobre mudanças de status, orçamentos prontos, etc. Este é um problema genérico, altamente padronizado e ideal para ser resolvido com uma ferramenta ou serviço de mercado .

2.13 Bounded Contexts Identificados(Contextos Delimitados)

Para garantir o máximo desacoplamento e a alta coesão do sistema, a solução TurboCat CatCar é decomposta nos seguintes Bounded Contexts. Cada contexto representa uma fronteira explícita em torno de um modelo de domínio consistente e alinhado a uma capacidade de negócio específica.

FrontOffice Context

- **Responsabilidade Principal:** Gerenciar toda a interação com o cliente, desde a criação da Ordem de Serviço até a aprovação do orçamento.
- **Tipo de Subdomínio:** Core Domain. A qualidade da interação neste contexto impacta diretamente a percepção de confiança do cliente.
- **Agregados Principais: `WorkOrder`, `Customer`, `Vehicle`.**
- **Descrição:** Este contexto é a porta de entrada para os serviços da oficina. Ele lida com o agendamento, o registro de informações do cliente e do veículo, a criação da Ordem de Serviço (`WorkOrder`) e a comunicação do orçamento para aprovação.

Workshop Context

- **Responsabilidade Principal:** Orquestrar a execução do reparo físico do veículo e gerenciar os recursos técnicos.
- **Tipo de Subdomínio:** Core Domain. A eficiência e a qualidade do trabalho realizado aqui são o cerne do serviço prestado.
- **Agregados Principais: `RepairJob`, `Mechanic`.**
- **Descrição:** É o coração da operação. Uma vez que uma `WorkOrder` é aprovada, o Workshop Context assume, criando `RepairJobs` específicos, alocando mecânicos, registrando o progresso técnico e emitindo as atualizações de status que serão consumidas por outros contextos.

Inventory Context

- **Responsabilidade** Principal: Controlar o estoque de peças, gerenciar fornecedores e o processo de aquisição de novos itens.
- **Tipo de Subdomínio:** Supporting Subdomain. É essencial para o negócio, mas não é o diferencial competitivo principal.
- **Agregados Principais: `InventoryItem`, `PurchaseOrder`, `Supplier`.**
- **Descrição:** Gerencia a disponibilidade, o custo e a localização das peças no estoque. Expõe uma API (Open-Host Service) para que outros contextos, principalmente o Workshop, possam consultar e requisitar itens necessários para os reparos.

Billing Context

- **Responsabilidade Principal:** Lidar com todo o processo de faturamento, cobrança e registro de pagamentos.

- **Tipo de Subdomínio:** Supporting Subdomain**.**
- **Agregados Principais: Invoice, Payment.**
- **Descrição:** Ativado por eventos como a aprovação de um orçamento ou a conclusão de um serviço, este contexto é responsável por gerar as faturas, processar os pagamentos e manter um registro financeiro preciso de cada Ordem de Serviço.

Notifications Context

- **Responsabilidade Principal:** Gerenciar o envio de todas as comunicações transacionais externas para os clientes.
- **Tipo de Subdomínio:** Generic Subdomain.
- **Agregados Principais: NotificationRequest.**
- **Descrição:** Atua como um serviço centralizado e desacoplado para o envio de notificações (e.g., e-mails, SMS). Ele consome eventos de outros contextos (como uma mudança de status no Workshop) e os traduz em comunicações claras para o cliente, utilizando serviços externos.

2.14 Mapeamento Subdomians(problema) x Bounded Contexts(solução)

Subdomínio (Problema)	Bounded Context (Solução)
Gerenciamento do Ciclo de Vida do Reparo	FrontOffice Context + Workshop Context
Gestão de Inventário e Aquisições	Inventory Context
Gestão de Clientes e Veículos	FrontOffice Context
Faturamento e Pagamentos	Billing Context
Comunicações Transacionais	Notifications Context

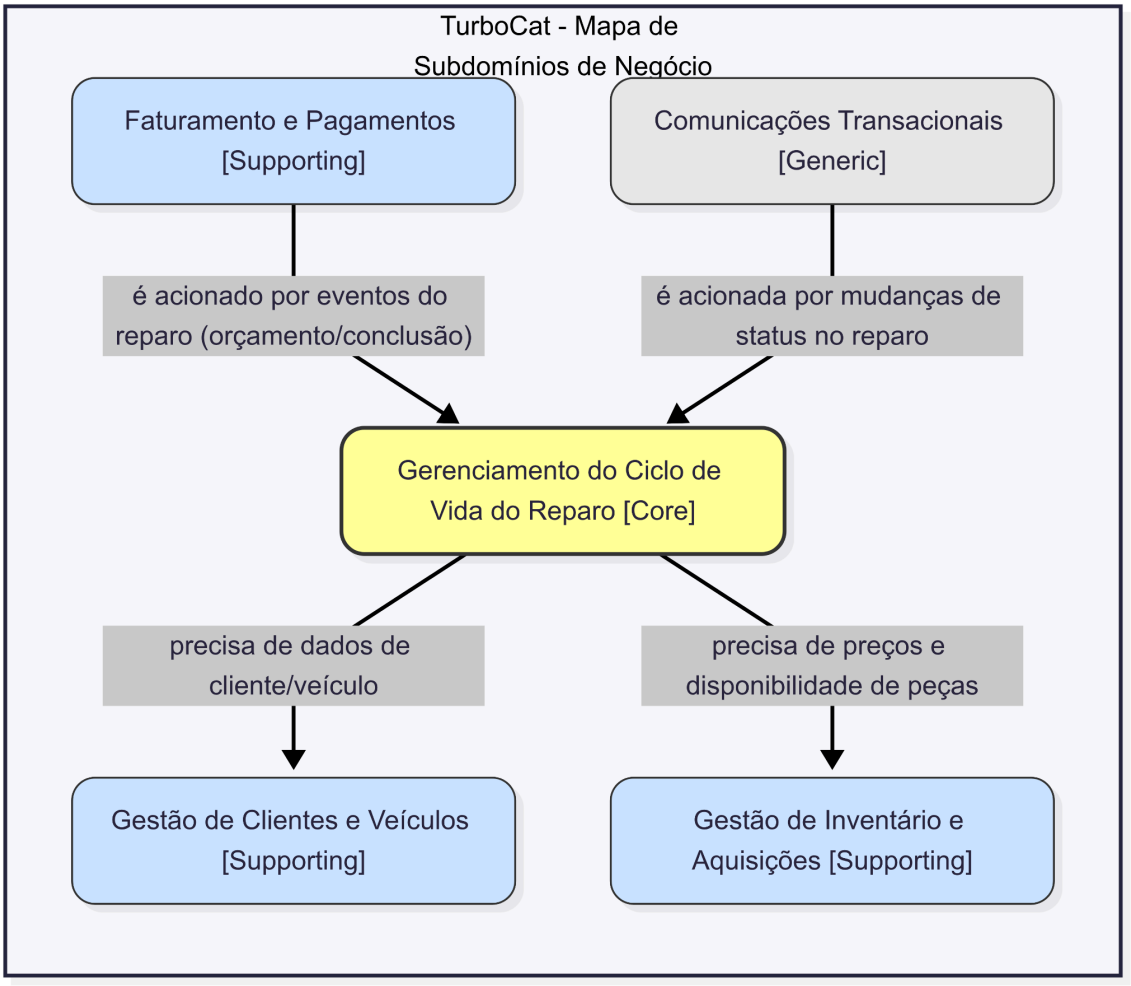
O Core Domain é tão complexo que foi dividido em dois Bounded Contexts (FrontOffice e Workshop) para gerenciar melhor as responsabilidades. Por outro lado, o FrontOffice Context absorveu a responsabilidade do subdomínio de suporte de "Gestão de Clientes" por uma questão de coesão do fluxo de trabalho.

2.15 Mapa visuais dos contextos

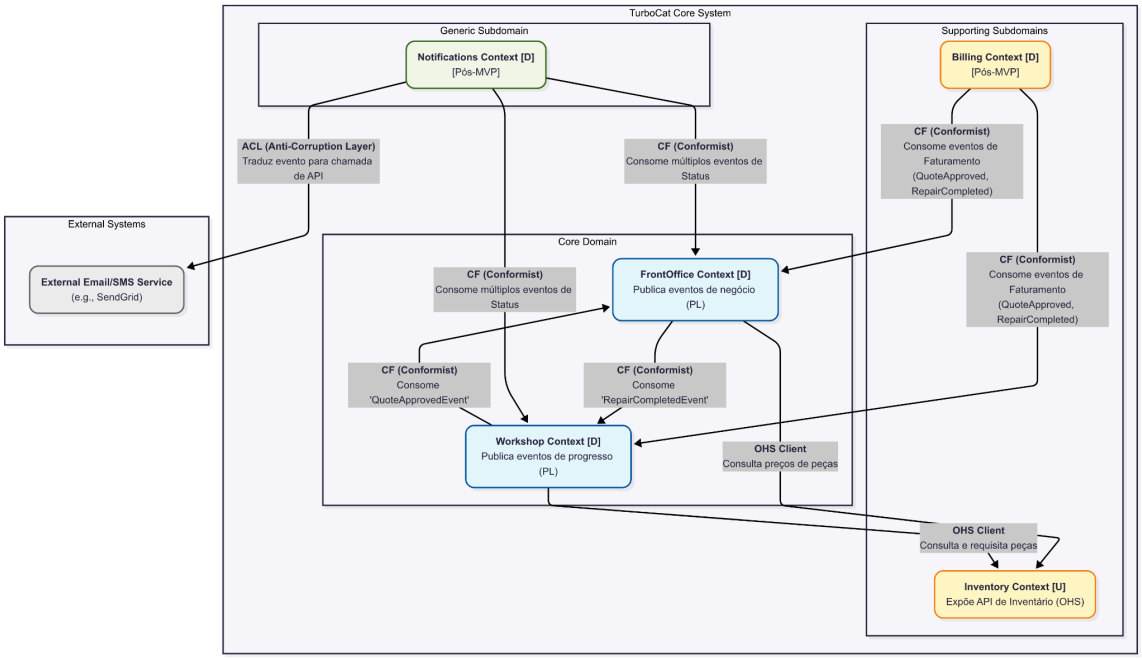
2.15.1 Visão de Subdomínios (O Espaço do Problema)

Propósito: Este mapa foca nas áreas de negócio e em como elas dependem umas das outras. Ele ignora a tecnologia e mostra as relações de poder e influência entre os

domínios. A seta --> indica uma dependência, onde um domínio precisa de informações ou é acionado por outro.



2.15.2 Visão de Bounded Contexts (O Espaço da Solução)
Propósito: Este mapa mostra os módulos de software (Bounded Contexts) e os padrões de integração técnica entre eles. A notação [U] significa Upstream (fornecedor do modelo/API) e [D] significa Downstream (consumidor, que se conforma ao modelo do Upstream). As setas apontam do Downstream para o Upstream.



2.16 Dicionário da Linguagem Ubíqua

Este dicionário é a fonte única da verdade para os termos de negócio e seus equivalentes técnicos. Ele está organizado em categorias para facilitar a consulta e garantir que toda a equipe — negócio, desenvolvimento e IA — fale a mesma língua.

❖ Atores e Papéis Principais (Actors & Roles)

Termo (Negócio)	Term (Código)	Definição no Contexto
Cliente	`Customer`	Pessoa ou empresa proprietária de veículos que solicita os serviços. É uma Raiz de Agregado.
Administrativo/Gerente	`AdministrativeStaff` (Role)	Usuário interno do sistema responsável por criar a OS, preparar orçamentos e se comunicar com o cliente. Não é um agregado, mas um papel de usuário.

Mecânico	Mechanic	Recurso técnico da oficina responsável pela execução do reparo. É uma Raiz de Agregado no Workshop Context.

❖ **Artefatos e Documentos Chave (Key Artifacts)**

Termo (Negócio)	Term (Código)	Definição no Contexto	Contexto Principal
Ordem de Serviço (OS)	WorkOrder	O agregado central que representa o **contrato com o cliente**. Rastreia o ciclo de vida completo do serviço sob a perspectiva do cliente.	FrontOffice
Orçamento	Quote	**Componente** da WorkOrder, não um agregado separado. Representa a lista de peças e serviços (QuoteLine) e seu estado (CustomerApproval).	FrontOffice
Linha do Orçamento	QuoteLine	Objeto de Valor (VO) dentro da WorkOrder que detalha um único item (peça ou serviço) a ser cobrado.	FrontOffice
Reparo / Serviço Técnico	RepairJob	O agregado que representa a **execução técnica interna** do trabalho, criado após a aprovação do Quote.	Workshop

Peça / Item de Estoque	InventoryItem	Um **item de catálogo** que representa um tipo de peça ou produto. Gerencia o preço e o nível de estoque (StockLevel).	Inventory

❖ **Processos e Verbos do Domínio (Domain Processes & Verbs)**

Termo (Negócio)	Comando	Descrição do Processo
Criar Ordem de Serviço	CreateWorkOrder	Inicia um novo fluxo de reparo, associando um cliente e um veículo a uma nova OS.
Preparar Orçamento	AddQuoteLine	Adicionar itens (peças e serviços) a uma WorkOrder para compor o orçamento.
Aprovar Orçamento	ApproveQuote	Ação do cliente que marca um orçamento como aprovado, permitindo o início do RepairJob.
Designar Mecânico	AssignMechanic	Alocar um mecânico disponível e qualificado a um RepairJob específico.
Iniciar Reparo	StartRepair	Mudar o status de um RepairJob para "Em Execução", indicando que o trabalho ativo começou.

Termo (Negócio)	Comando	Descrição do Processo
Criar Ordem de Serviço	CreateWorkOrder	Inicia um novo fluxo de reparo, associando um cliente e um veículo a uma nova OS.
Ajustar Estoque	AdjustStock	Aumentar ou diminuir a contagem de um InventoryItem (ex: após uma venda ou recebimento).

❖ **Estados e Ciclos de Vida (States & Lifecycles)**

Conceito (Negócio)	Term (Código / Enum)	Valores e Significado
Status da OS (Visível ao Cliente)	WorkOrderStatus	Received, InDiagnosis, AwaitingApproval, InExecution, Completed, Delivered. Reflete as macro-fases do serviço.
Status do Reparo (Interno)	RepairJobStatus	Pending, InProgress, AwaitingParts, Paused, QualityCheck, Finished. Detalha o estado operacional do trabalho na oficina.
Disponibilidade do Mecânico	MechanicAvailability	Available, OnJob, OnLeave. Controla se um mecânico pode ser alocado.

❖ **Conceitos e Qualificadores (Concepts & Qualifiers)**

Conceito (Negócio)	Implementação no Modelo	Descrição
Serviço de Confiança	(Diretriz Arquitetural)	O princípio de negócio que guia o design. Manifesta-se através da transparência de status (`WorkOrderStatus`) e da precisão do `Quote`.
Peça Fornecida pelo Cliente	Flag/Atributo na `QuoteLine`	Uma regra de negócio que indica que um item no orçamento não deve ser retirado do estoque nem cobrado. O agregado `WorkOrder` deve saber lidar com isso.
Transparência	(Atributo de Qualidade)	A capacidade do sistema de fornecer informações de status claras e atualizadas, tanto para clientes (`WorkOrderStatus`) quanto para gerentes (`RepairJobStatus`).

3. DESIGN TÁTICO

3.1 Exploração Preliminar do Modelo Tático

3.1.1 Blueprint dos Agregados, Entidades e VOs

Este blueprint detalha os Agregados para cada Bounded Context, explicitando seus atributos, regras de negócio (invariantes), e o comportamento esperado através de Comandos e Eventos de Domínio.

Contexto: FrontOffice

- **WorkOrder (Aggregate Root):** Orquestra o ciclo de vida da solicitação de serviço, desde a criação até a aprovação do orçamento pelo cliente.

    - **ATRIBUTOS:** WorkOrderId, CustomerId, VehicleId, Status (enum), List<QuoteLine>, CustomerApproval.
    - **COMPONENTES:**
        - **QuoteLine (Objeto de Valor):** Lista de linhas do orçamento (ItemId, Description, Quantity, UnitPrice, Type).
        - **CustomerApproval (Objeto de Valor):** Detalhes da aprovação (ApprovedBy, ApprovalTimestamp, ApprovalMethod).
    - **INVARIANTES:**
        - Uma WorkOrder deve sempre estar associada a um Customer e um Vehicle existentes.
        - Itens do orçamento (QuoteLines) só podem ser adicionados/removidos antes do orçamento ser enviado ao cliente.
        - A aprovação do cliente só pode ser registrada após o orçamento ser enviado.
        - As transições de Status devem seguir um fluxo predefinido (ex: Recebida -> Em Diagnóstico -> Aguardando Aprovação).
    - **COMMANDOS:** CreateWorkOrder, AddQuoteLine, SendQuoteForApproval, ApproveQuote, RejectQuote.
    - **EVENTOS:** WorkOrderCreated, QuoteLineAdded, QuoteSentForApproval, QuoteApprovedByCustomer, QuoteRejectedByCustomer.
- **Customer (Aggregate Root):** Gerencia os dados cadastrais do cliente e a lista de seus veículos.
    - **ATRIBUTOS:** CustomerId, Name, ContactInfo, DocumentId, List<Vehicle>.
    - **COMPONENTES:**
        - **Vehicle (Entity):** Lista de veículos do cliente.
        - **ContactInfo (Value Object):** Dados de contato (Email, PhoneNumber).
        - **DocumentId (Value Object):** Documento de identificação (Number, Type).
    - **INVARIANTES:**
        - O DocumentId (CPF/CNPJ) deve ser único no sistema.
        - Um veículo não pode ser adicionado duas vezes ao mesmo cliente.
    - **COMMANDOS:** RegisterNewCustomer, UpdateContactInfo, AddVehicleToCustomer.
    - **EVENTOS:** CustomerRegistered, CustomerContactInfoUpdated, VehicleAddedToCustomer.
- **Vehicle (Entity):** Representa um veículo específico de um cliente. Não é uma raiz de agregado.
    - **ATRIBUTOS:** VehicleId, LicensePlate, Model, Year, Color.
    - **INVARIANTES:** Suas regras são garantidas pelo agregado Customer.

Contexto: Workshop

- **RepairJob (Aggregate Root):** Representa a execução do trabalho técnico na oficina após a aprovação do orçamento.
    - **ATRIBUTOS:** RepairJobId, WorkOrderId, Status (enum), List<TechnicalNote>,

AssignedMechanicId.
    - **COMPONENTES:**
        - **TechnicalNote (Value Object):** Lista de anotações técnicas (NoteText, AuthorId, Timestamp)**.**
    - **INVARIANTES:**
        - Só pode ser criado a partir de uma WorkOrder com o orçamento aprovado.
        - Um mecânico deve ser designado antes que o Status mude para "Em Execução".
    - **COMMANDOS:** CreateRepairJob, AssignMechanic, StartRepair, AddTechnicalNote, CompleteRepair.
    - **EVENTOS:** RepairJobCreated, MechanicAssignedToJob, RepairStarted, TechnicalNoteAdded, RepairCompleted.
- **Mechanic (Aggregate Root):** Representa o mecânico como um recurso da oficina.
    - **ATRIBUTOS:** MechanicId, Name, Specializations (List), CurrentAvailability (enum).
    - **COMPONENTES:**
        - **Specializations (List<string>):** Lista de especialidades.
    - **INVARIANTES:** A disponibilidade deve ser gerenciada de forma consistente.
    - **COMMANDOS:** RegisterNewMechanic, UpdateAvailability, AddSpecialization.
    - **EVENTOS:** MechanicRegistered, MechanicAvailabilityUpdated.

Contexto: Inventory

- **InventoryItem (Aggregate Root):** Gerencia uma peça ou insumo específico no estoque.
    - **ATRIBUTOS:** PartId, Name, Description, Price, StockLevel.
    - **COMPONENTES:**
        - **Price (Value Object):** Preço da peça (Amount, Currency).
        - **StockLevel (Value Object):** Quantidade em estoque (Quantity, UnitOfMeasure).
    -
    - **INVARIANTES:** StockLevel não pode ser negativo. O preço não pode ser negativo.
    - **COMMANDOS:** RegisterNewPart, AdjustStock, UpdatePrice.
    - **EVENTOS:** NewPartRegistered, StockAdjusted, PartPriceUpdated.

Contexto: Billing (Pós-MVP)

- **Invoice (Aggregate Root):** Representa uma fatura financeira a ser paga pelo cliente.
    - **ATRIBUTOS:** InvoiceId, WorkOrderId, TotalAmount, Status (enum), DueDate.
    - **INVARIANTES:** O valor total deve corresponder ao orçamento aprovado. Não pode ser alterada após ser paga.
    - **COMMANDOS:** GenerateInvoice, MarkAsPaid, MarkAsOverdue.
    - **EVENTOS:** InvoiceGenerated, InvoicePaid, InvoiceMarkedAsOverdue.

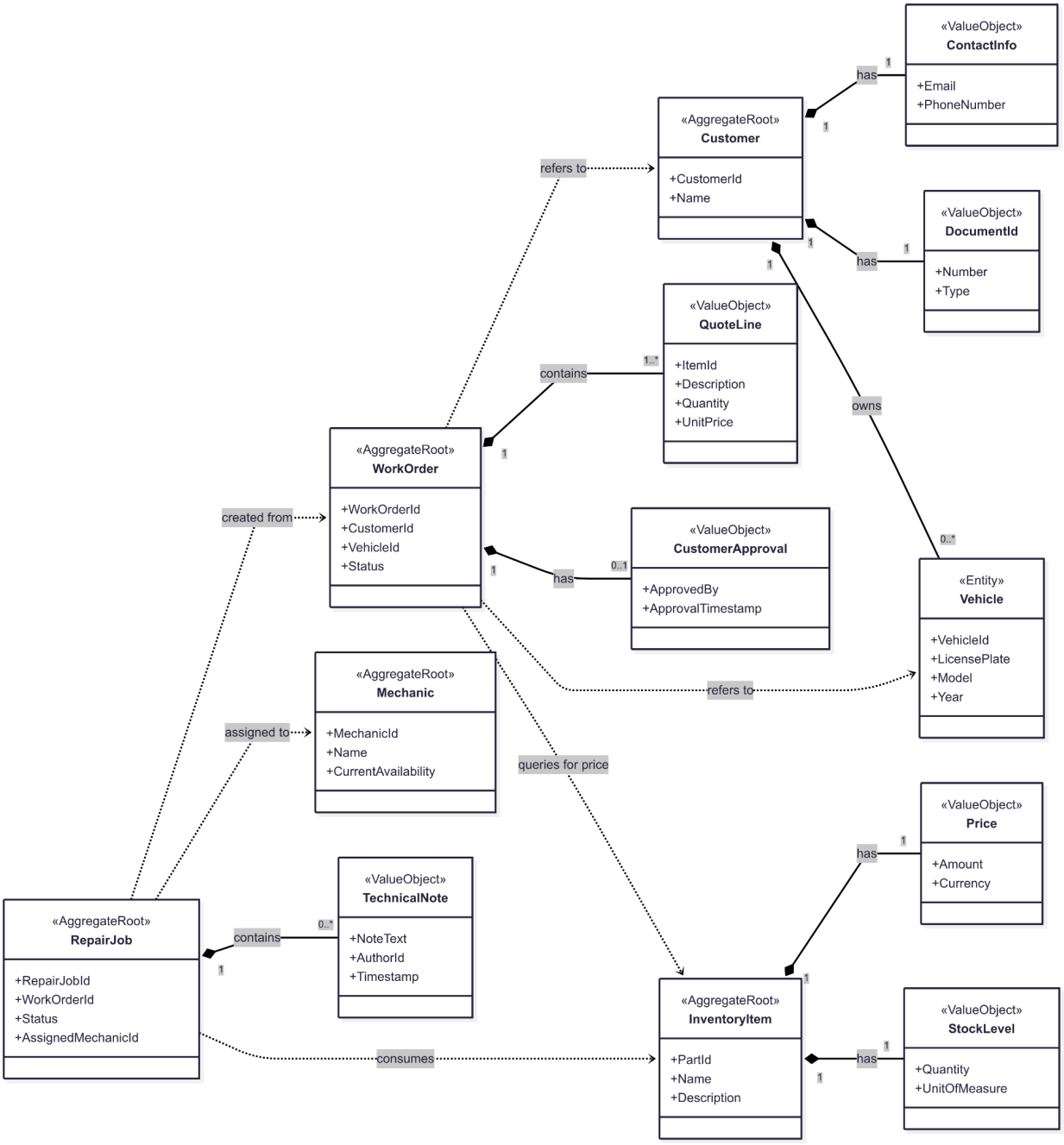
- **Payment (Aggregate Root):** Representa um pagamento efetuado.
    - **ATRIBUTOS:** PaymentId, InvoiceId, AmountPaid, PaymentMethod (enum), PaymentTimestamp.
    - **INVARIANTES:** O valor pago não pode exceder o valor devido da fatura.
    - **COMMANDOS:** RecordPayment.
    - **EVENTOS:** PaymentRecorded.

Contexto: Notifications (Pós-MVP)

- **NotificationRequest (Aggregate Root):** Gerencia uma única solicitação de notificação a ser enviada.
    - **ATRIBUTOS:** NotificationId, Recipient, Channel (Email/SMS), Message, Status (enum).
    - **INVARIANTES:** Deve ter um destinatário e uma mensagem válidos.
    - **COMMANDOS:** SendNotification.
    - **EVENTOS:** NotificationSent, NotificationFailed.

3.1.2 Diagrama de Classes dos Agregados

O diagrama a seguir representa visualmente a estrutura detalhada dos Agregados, Entidades e Objetos de Valor para cada Bounded Context, incluindo seus atributos principais e as relações entre eles.



3.2 Formalização dos Contratos de Dados

Os contratos de dados são essenciais para definir as fronteiras de comunicação do sistema e são categorizados em três tipos principais:

3.2.1 Diretriz Geral de Serialização

- **Comunicação Externa (API para Clientes):** Utiliza **JSON**, por ser o padrão web, legível e universalmente suportado.

- **Comunicação Interna (Entre Bounded Contexts):** Prioriza **Protobuf (Protocol Buffers)** para troca de eventos via Message Bus, devido à sua natureza binária, performance e tipagem forte, ideal para comunicação entre serviços.

3.2.2 Contratos de API Externa (JSON DTOs)

São os DTOs (Data Transfer Objects) que estruturam os dados para requisições (input) e respostas (output) da API RESTful.

Exemplo de Input (Comando): POST /v1/work-orders
Payload para criar uma nova Ordem de Serviço.

```
{
  "customerId": "c4b5c6d7-1b3a-4e9a-9b1e-0a1b2c3d4e5f",
  "vehicleId": "a1b2c3d4-5e6f-7a8b-9c0d-1e2f3a4b5c6d",
  "problemDescription": "Veículo não liga e faz barulho estranho ao girar a chave."
}
```

Exemplo de Output (Query): GET /v1/work-orders/{id}/status
Resposta simplificada para consulta pública de status de uma OS.

```
{
  "workOrderId": "f4a5b6c7-8d9e-0f1a-2b3c-4d5e6f7a8b9c",
  "vehicleDescription": "Toyota Corolla 2022 (ABC-1234)",
  "currentStatus": "AwaitingApproval",
  "statusDescription": "Orçamento enviado. Aguardando aprovação do cliente.",
  "lastUpdate": "2025-07-23T14:30:00Z"
}
```

3.2.3 Contratos de Eventos Internos (Protobuf)

São eventos de domínio publicados no Message Bus para comunicação assíncrona entre Bounded Contexts.

Exemplo: QuoteApprovedByCustomerEvent.proto
Evento publicado pelo FrontOffice Contex` quando um orçamento é aprovado, para ser consumido pelo Workshop Context.

```
syntax = "proto3";

package turbocat.events.v1;

// Evento disparado quando um cliente aprova um orçamento.
message QuoteApprovedByCustomerEvent {
  string work_order_id = 1;
```

```
  string customer_id = 2;

  // Informações básicas do veículo para contexto rápido.
  string vehicle_info = 3;

  // Lista de itens que foram aprovados para o reparo.
  repeated ApprovedItem approved_items = 4;
}

// Representa um item individual aprovado no orçamento.
message ApprovedItem {
  string item_id = 1;
  string description = 2;
  int32 quantity = 3;
  ItemType type = 4; // Distingue entre PEÇA e SERVIÇO.
}

enum ItemType {
  ITEM_TYPE_UNSPECIFIED = 0;
  PART = 1;
  SERVICE = 2;
}
```

3.2.4 Contratos de Leitura (Query DTOs)
São DTOs otimizados para leitura, desnormalizando dados de múltiplos agregados ou contextos para servir uma tela ou consulta específica de forma eficiente (princípio do CQRS).

Exemplo: `WorkOrderDetailsDto.cs`
DTO retornado por `GET /v1/work-orders/{id}` para a visão administrativa, contendo dados de múltiplos agregados.

```
public record WorkOrderDetailsDto
{
    // Dados do agregado WorkOrder (FrontOffice)
    public Guid WorkOrderId { get; init; }
    public string Status { get; init; }
    public string ProblemDescription { get; init; }
    public DateTime CreatedAt { get; init; }

    // Dados do agregado Customer (FrontOffice)
    public Guid CustomerId { get; init; }
    public string CustomerName { get; init; }
    public string CustomerContactPhone { get; init; }

    // Dados da entidade Vehicle (FrontOffice)
    public Guid VehicleId { get; init; }
    public string VehicleLicensePlate { get; init; }
    public string VehicleModel { get; init; }
```

```
    // Lista de linhas do orçamento (VO do WorkOrder)
    public IReadOnlyList<QuoteLineDto> QuoteLines { get; init; } = [];

    // Detalhes da aprovação (VO do WorkOrder)
    public CustomerApprovalDto? ApprovalDetails { get; init; }
}

public record QuoteLineDto(string Description, int Quantity, decimal UnitPrice, string
Type);
public record CustomerApprovalDto(string ApprovedBy, DateTime ApprovalTimestamp);
```

3.3 Design de Integração e Resiliência

Esta seção detalha as estratégias de comunicação e tolerância a falhas para os principais pontos de integração entre os Bounded Contexts, conforme mapeado na seção 2.10.

Ficha de Integração #1: Front Office [U] -> Workshop [D]

- **Ponto de Integração:** Comunicação da aprovação de um orçamento para o início do trabalho.
- **Gatilho de Negócio:** O cliente aprova o orçamento no FrontOffice Context.
- **Padrão de Integração: Publish/Subscribe (Pub/Sub)**. O FrontOffice publica um evento QuoteApprovedByCustomerEvent. O Workshop é um assinante (subscriber) deste evento.
- **Estratégia de Resiliência:**
    - **Na Publicação (FrontOffice):** Será utilizado o **Outbox Pattern**. A gravação da aprovação no banco de dados e a criação da mensagem do evento são feitas em uma única transação atômica. Um processo em segundo plano (ou o próprio Wolverine, que tem suporte nativo para isso) garante a entrega da mensagem ao broker, mesmo que o broker esteja temporariamente indisponível no momento da transação.
    - **No Consumo (Workshop):** O handler do evento será idempotente. Em caso de falha no processamento, serão aplicadas **retentativas com backoff exponencial**. Se as falhas persistirem, a mensagem será movida para uma **Dead-Letter Queue (DLQ)** para análise manual, evitando o bloqueio da fila principal.
- **Justificativa:** O desacoplamento é máximo. O FrontOffice não precisa saber ou se importar com o que acontece após a aprovação, apenas garante que o evento seja publicado.

Ficha de Integração #2: Workshop [D] -> Inventory [U]

- **Ponto de Integração:** Consulta e reserva de peças para um reparo.
- **Gatilho de Negócio:** Um mecânico no Workshop precisa verificar a disponibilidade ou

requisitar uma peça para um RepairJob.

- **Padrão de Integração: Open-Host Service (OHS)**. O Inventory Context expõe uma API síncrona (RESTful/gRPC) que o Workshop consome como um cliente.
- **Estratégia de Resiliência:**
    - **No Cliente (Workshop):** Será implementado o padrão **Circuit Breaker**. Se a API do Inventory começar a falhar ou ficar lenta, o circuito "abre" e as chamadas subsequentes falham imediatamente por um período, protegendo o Workshop de ficar travado.
    - **Timeouts Agressivos:** As chamadas à API terão um timeout curto (ex: 500ms) para evitar longas esperas.
    - **Fallback:** Em caso de falha na comunicação, o sistema deve registrar a falha e potencialmente mover o RepairJob para um status de "Aguardando Peça (Sistema de Inventário Indisponível)", notificando a equipe administrativa.
- **Justificativa:** A necessidade de dados é em tempo real. O Workshop precisa de uma resposta imediata sobre a disponibilidade da peça para prosseguir. A comunicação assíncrona não seria adequada para este caso de uso.

Ficha de Integração #3: Front Office [D] -> Inventory [U]

- **Ponto de Integração:** Consulta de preços de peças para montagem de um orçamento.
- **Gatilho de Negócio:** Um funcionário administrativo no FrontOffice adiciona uma peça a uma WorkOrder.
- **Padrão de Integração: Open-Host Service (OHS)**, consumindo a mesma API do Inventory Context.
- **Estratégia de Resiliência:** Idêntica à da Ficha #2 (**Circuit Breaker** e **Timeouts Agressivos**). A falha em obter um preço não pode impedir a operação do FrontOffice. O sistema pode, como fallback, permitir a inserção manual do preço.
- **Justificativa:** Necessidade de dados de preços em tempo real para garantir a precisão do orçamento.

3.4 Tecnologias e Stack de Desenvolvimento

- **Linguagem:** C# com .NET 9.
- **Framework Principal:** .NET 9 com .NET Aspire para orquestração de desenvolvimento.
- **Command Bus / Mensageria:** Wolverine.
- **Tratamento de Resultados:** RiseOn.ResultRail.
- **Validação:** FluentValidation.
- **Banco de Dados:** PostgreSQL (gerenciado via .NET Aspire).
- **ORM:** Entity Framework Core.

3.5 Blueprint Arquitetural do Bounded Context e Estrutura da Solução

- **Padrão Escolhido:** Arquitetura Híbrida: **Vertical Slice + CQRS com Command Bus**.
- **Justificativa:** Combina o isolamento de domínio da Clean Architecture com a alta coesão da organização por features (Vertical Slice), ideal para a equipe e para a evolução futura.

3.4.1. Visão Geral da Arquitetura

O sistema é um **Monólito Modular** com cada **Bounded Context (BC)** existindo como um projeto .csproj independente.

- **Ponto de Entrada Único:** Um projeto TurboCat.Api serve como a fachada RESTful.
- **Orquestração de Desenvolvimento:** O projeto TurboCat.AppHost (.NET Aspire) gerencia o ambiente de desenvolvimento.
- **Command Bus e Mensageria:** O padrão **Command Bus** é implementado com a biblioteca **Wolverine**. Wolverine gerencia a execução dos Use Cases, agindo como um "mediador super-poderoso". Ele lida com a descoberta de handlers, execução in-process e fornece um caminho claro para a evolução para mensageria assíncrona (com RabbitMQ/Kafka) e o padrão Outbox, sem a necessidade de alterar a lógica de negócio.
- **Tratamento de Resultados:** Todas as operações que podem falhar (praticamente todos os Use Cases) devem retornar um objeto Result da biblioteca **RiseOn.ResultRail**. Isso padroniza o tratamento de sucesso e erro em toda a aplicação, eliminando a necessidade de exceções para controle de fluxo.
- **Use Cases como Features (Vertical Slices):** Cada funcionalidade é uma "fatia vertical" coesa, implementada como um handler do Wolverine.

3.4.2. Estrutura de Diretórios e Projetos

A estrutura de diretórios permanece a mesma, pois é agnóstica às bibliotecas de implementação, o que demonstra sua robustez.

```
 /TurboCat.CatCar.sln
|
// ==================== PROJETOS DE ORQUESTRAÇÃO E ENTRADA
====================
|-- src/
|   |-- TurboCat.AppHost/
|   |-- TurboCat.ServiceDefaults/
|   |-- TurboCat.Api/
|       |-- Program.cs                  // Ponto de entrada.
Configura Wolverine, Auth, DB.
|       |-- Endpoints/
|           |-- WorkOrderEndpoints.cs   // Minimal API que envoca os
comandos Wolverine.
|
// ==================== PROJETOS DE BOUNDED CONTEXT
====================
|   |-- BoundedContexts/
|       |-- TurboCat.FrontOffice/       // Projeto do Bounded
Context "Front Office"
|           |-- Domain/                 // CORE DOMAIN: Agregados,
VOs, Eventos, Interfaces de Repositório.
```

```
|           |-- Features/               // USE CASES / APPLICATION
LAYER: Handlers do Wolverine.
|           |   |-- CreateWorkOrder/
|           |       |-- CreateWorkOrder.cs  // Arquivo único com
Command, Handler e Validator.
|           |-- Infrastructure/         // IMPLEMENTAÇÕES:
Repositórios EF Core, etc.
|
|       |-- TurboCat.Workshop/
|           |-- (mesma estrutura interna: Domain, Features,
Infrastructure)
|
// ================== PROJETOS DE TESTE ==================
|-- tests/
    |-- ... (Estrutura de testes como antes)
```

3.4.3. Anatomia de uma Feature

Um Use Case é um handler. A abordagem de **um único arquivo por feature** é ideal.

Exemplo de arquivo:
src/BoundedContexts/TurboCat.FrontOffice/Features/CreateWorkOrder.cs

```
using FluentValidation;
using RiseOn.ResultRail;
using Wolverine;

namespace TurboCat.FrontOffice.Features.CreateWorkOrder;

// 1. INPUT (O Comando/Mensagem): Um simples record. Não precisa de
interfaces.
public record Command(Guid CustomerId, Guid VehicleId, string
ProblemDescription);

// 2. ORQUESTRADOR (O Handler): Um método público que consome o
comando.
public static class Handler
{
        public static async Task<Result<Guid>> Handle(
        Command command,
        IWorkOrderRepository workOrderRepository)
    {
        // Passo A: Validação (será executada pelo middleware antes
deste método).
        // Passo B: Execução do Domínio. O Handler invoca a lógica
no Agregado.
        var workOrder = WorkOrder.Create(
            command.CustomerId,
            command.VehicleId,
            command.ProblemDescription);
```

```
        // Passo C: Persistência. O Handler usa a abstração do
repositório.
        await workOrderRepository.AddAsync(workOrder);

        // Passo D: Retorno com ResultRail. Retorna sucesso com o ID
da nova OS.
        return Result.Success(workOrder.Id.Value);
    }
}
// NOTA: Em cenários onde um erro de negócio é detectado, o retorno
seria:
// return Result.Failure<Guid>(new Error("WorkOrder.AlreadyExists",
"Uma OS já existe para este veículo."));

public class Validator : AbstractValidator<Command>
{
    public Validator()
    {
        RuleFor(x => x.CustomerId).NotEmpty();
        RuleFor(x => x.VehicleId).NotEmpty();
        RuleFor(x =>
x.ProblemDescription).NotEmpty().MaximumLength(500);
    }
}
```

3.4.4. Configuração na API (Program.cs)

```
// Em TurboCat.Api/Program.cs
      var builder = WebApplication.CreateBuilder(args);
// ... outros serviços (Swagger, Auth, etc.)
// Adiciona o Wolverine, que automaticamente escaneará os assemblies
// em busca de handlers. A política de descoberta é configurável.
// Também integramos o FluentValidation ao pipeline do Wolverine.
      builder.Host.UseWolverine(opts =>
      {
    // Integração com FluentValidation
    opts.UseFluentValidation();
            // Outras configurações(ex: Outbox, transportes) podem
ser adicionadas aqui.
      });
      var app = builder.Build();

// Exemplo de como um endpoint é invocado.
      app.MapPost("/v1/work-orders", async (CreateWorkOrder.Command
command, IMessageBus bus) =>
      {
            var result = await
bus.InvokeAsync<Result<Guid>>(command);
            // O helper "Handle" do ResultRail pode ser usado para
mapear o resultado para uma resposta HTTP.
            return result.Handle(
                  onSuccess: id =>
Results.Created($"/v1/work-orders/{id}", id),
                  onFailure: error => Results.BadRequest(error)
```

```
            );
    });

app.Run();
```

Este princípio fundamental **permanece inalterado e é o mais importante**:

- Infrastructure **depende de** Domain.
- Features (Application) **depende de** Domain.
- Domain **não depende de ninguém**.

3.6 Princípios de Aplicação Cloud-Native: The 12-Factor App

A solução TurboCat CatCat adere aos princípios do 12-Factor App para garantir que seja robusta, escalável e otimizada para ambientes de nuvem. O framework **.NET Aspire** foi escolhido em parte por sua aderência nativa a muitos desses fatores, simplificando a implementação. Abaixo, um paralelo entre os fatores mais relevantes e como o Aspire nos ajuda a cumpri-los.

I. Base de Código: Uma base de código, muitos *deploys*

- **Adesão:** O projeto reside em um único repositório Git. A arquitetura de Monólito Modular permite um único *deploy* para o MVP, mas o design em Bounded Contexts facilita a futura extração para múltiplos repositórios e *deploys* independentes, se necessário.

II. Dependências: Isole e declare as dependências explicitamente

- **Adesão:** O .NET utiliza o sistema de pacotes NuGet e arquivos .csproj para declarar explicitamente todas as dependências do projeto. Isso garante que o ambiente seja reprodutível em qualquer máquina.

III. Configuração: Armazene a configuração no ambiente

Adesão com Aspire: Este é um ponto forte do Aspire. O AppHost injeta automaticamente configurações, como *connection strings*, através de variáveis de ambiente. Isso elimina segredos e configurações específicas de ambiente do código-fonte.

// O Aspire injeta a connection string como uma variável de ambiente no 'api'.

```
var postgres = builder.AddPostgres("db").AddDatabase("turbocatdb");
var api =
builder.AddProject<Projects.TurboCat_Api>("api").WithReference(postg
res);
```

IV. Serviços de Apoio: Trate os serviços de apoio como recursos acoplados

- **Adesão com Aspire:** O Aspire modela explicitamente os serviços de apoio (PostgreSQL, Redis, etc.). Suas informações de conexão são fornecidas à aplicação via configuração (Fator III), tratando-os como recursos externos acessíveis por uma

URL/credenciais.

V. Build, release, run: Separe estritamente os estágios

- **Adesão:** O processo seguirá um pipeline de CI/CD padrão. O estágio de ***build*** compila o código e suas dependências. O estágio de *release* combina o artefato do build com a configuração do ambiente de destino. O estágio de ***run*** executa a aplicação no ambiente.

VI. Processos: Execute a aplicação como um ou mais processos *stateless*

- **Adesão:** A API RESTful é projetada para ser *stateless*. Qualquer estado necessário é persistido em um serviço de apoio (o banco de dados PostgreSQL), permitindo que múltiplas instâncias do processo da API sejam executadas em paralelo.

VII. Vínculo de Porta: Exponha serviços via portas

- **Adesão com Aspire:** O Aspire gerencia automaticamente o vínculo de portas para cada serviço durante o desenvolvimento local, evitando conflitos e simplificando a descoberta de serviços entre os componentes da aplicação.

VIII. Concorrência: Escale através do modelo de processos

- **Adesão:** Sendo *stateless* (Fator VI), a aplicação pode ser escalada horizontalmente simplesmente adicionando mais processos (contêineres). O modelo de processos do sistema operacional gerencia a concorrência.

IX. Descartabilidade: Maximize a robustez com inicialização rápida e desligamento gracioso

- **Adesão com Aspire:** Ao rodar os componentes em contêineres, o Aspire promove a descartabilidade. É fácil iniciar, parar e reiniciar qualquer parte do sistema. Os *Health Checks*, facilitados pelo Aspire, ajudam o orquestrador a saber quando um serviço está pronto ou precisa ser reiniciado.

X. Paridade Dev/Prod: Mantenha os ambientes o mais semelhante possível

- **Adesão com Aspire:** O uso de contêineres Docker para os serviços de apoio (como PostgreSQL) no AppHost garante que o ambiente de desenvolvimento seja extremamente similar ao de produção, que também usará contêineres, reduzindo a chance de *bugs* que "só acontecem em produção".

XI. Logs: Trate *logs* como fluxos de eventos

- **Adesão com Aspire:** O Aspire fornece um **painel de controle (*dashboard*) unificado** que agrega *logs*, *traces* e métricas de todos os componentes em um único local. A aplicação escreve para a saída padrão (ILogger), e o Aspire cuida da coleta e da

apresentação, cumprindo perfeitamente este fator.

XII. Processos de Administração: Execute tarefas de *admin* como processos *one-off*

- **Adesão com Aspire:** Tarefas como migrações de banco de dados (EF Core Migrations) podem ser configuradas como projetos executáveis (Executable) no AppHost, garantindo que elas rodem como processos separados que compartilham a mesma base de código e configuração da aplicação principal.

Ao adotar o .NET Aspire, a equipe pode focar na lógica de negócio, confiando que o framework já fornece as fundações para uma aplicação Cloud-Native que segue os princípios do 12-Factor App.

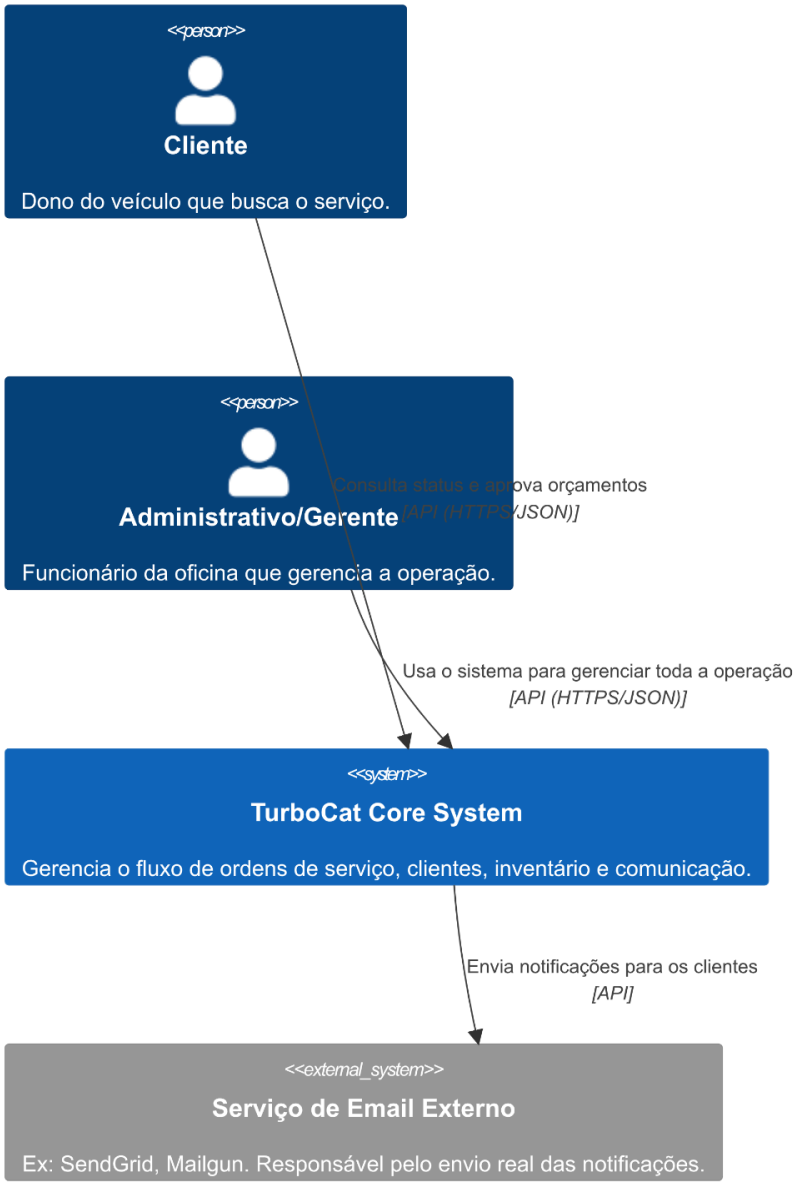
4. C4 MODEL: VISUALIZANDO A ARQUITETURA

O C4 Model é utilizado para descrever a arquitetura do software em diferentes níveis de zoom. Cada nível atende a um público diferente e responde a perguntas específicas sobre a estrutura do sistema.

4.1 Nível 1: Diagrama de Contexto de Sistema

Propósito: Mostra a "grande foto". Descreve como o nosso sistema (`TurboCat Core System`) se encaixa no ambiente, interagindo com usuários e outros sistemas.
Público: Todos (equipe de negócio, desenvolvedores, stakeholders).

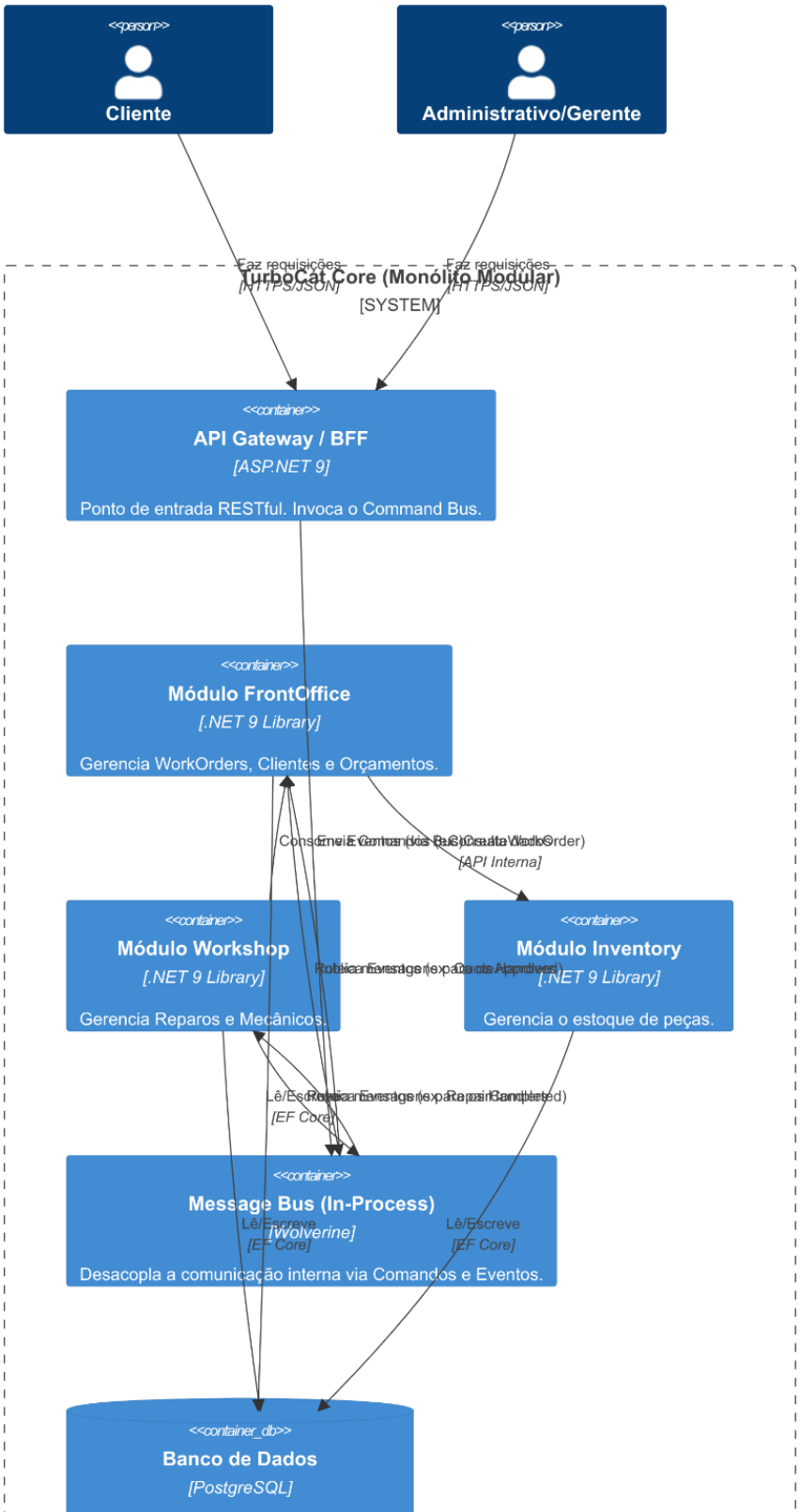
System Context for TurboCat CatCat



4.2 Nível 2: Diagrama de Contêineres

Propósito: Faz um "zoom" no nosso sistema, mostrando os principais blocos de construção de alto nível (os "contêineres", que podem ser aplicações, bancos de dados, bibliotecas, etc.) e como eles se comunicam.
Público: Arquitetos e Desenvolvedores.

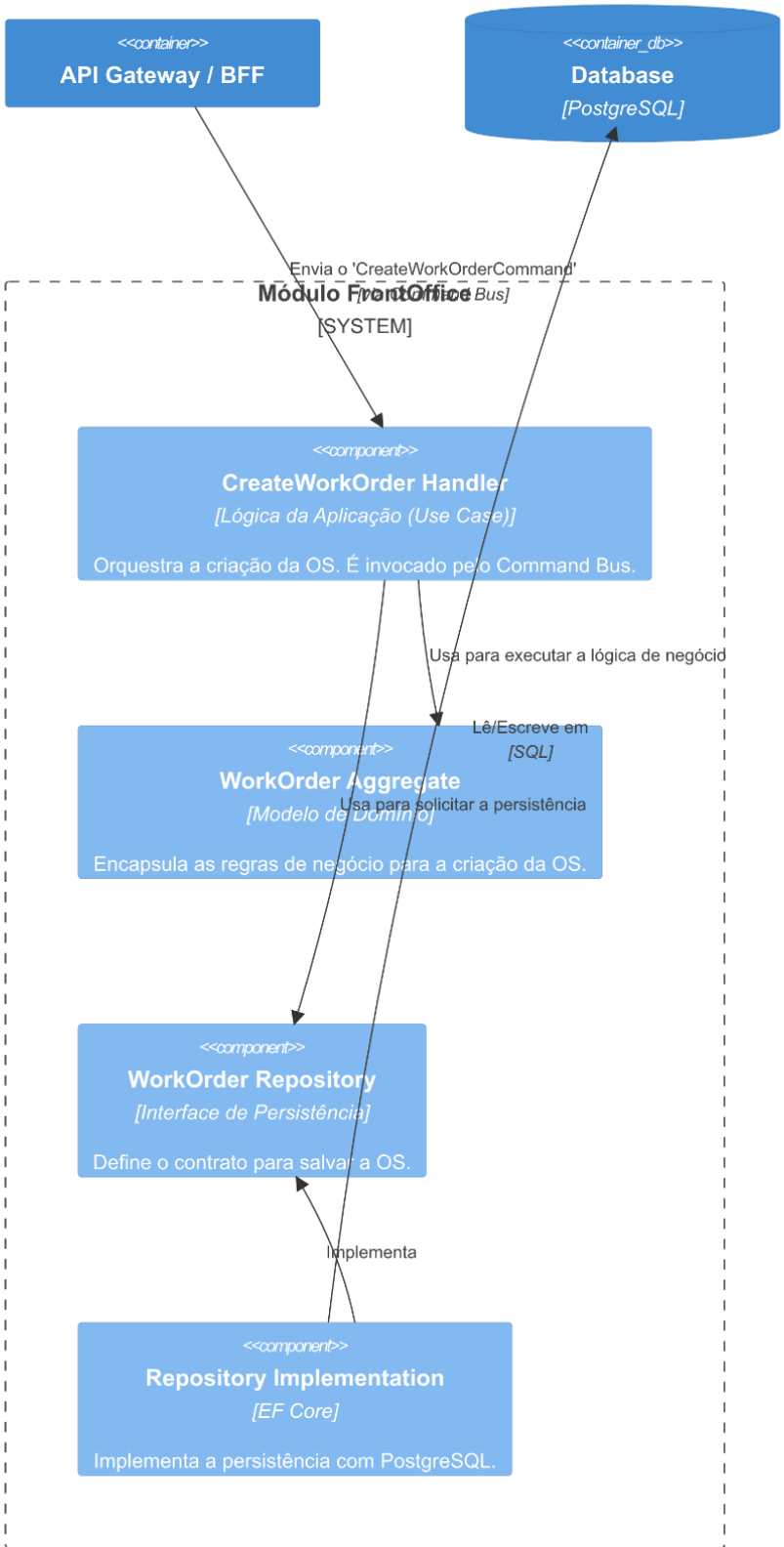
Container Diagram for TurboCat CatCar



4.3 Nível 3: Diagrama de Componentes (Foco no Módulo Front Office)

Propósito: Faz um "zoom" em um contêiner (neste caso, o Módulo FrontOffice) para mostrar seus principais componentes internos e como eles colaboram para realizar a funcionalidade "Criar Ordem de Serviço".
Público: Desenvolvedores que trabalham no módulo.

Component Diagram for Front Office Module (Feature: Create Work Order)



5. CONCLUSÃO

Este documento de design representa mais do que um simples blueprint técnico; ele é a consolidação de uma jornada estratégica, partindo da identificação do nosso **Core Domain** — o Gerenciamento do Ciclo de Vida do Reparo Confiável — até a definição de uma arquitetura pragmática e evolutiva.

A decisão de adotar um **Monólito Modular** foi deliberada, equilibrando a necessidade de agilidade para o MVP com uma visão de futuro para um sistema distribuído. As fronteiras entre os **Bounded Contexts** (FrontOffice, Workshop, Inventory) foram cuidadosamente mapeadas para garantir alta coesão e baixo acoplamento, preparando o terreno para uma manutenção e evolução sustentáveis.

No nível tático, a arquitetura de **Vertical Slices** com um **Command Bus** central garante que cada feature seja desenvolvida de forma isolada e testável. Os modelos de domínio, detalhados nos blueprints de **Agregados, Entidades e VOs**, são o coração do sistema, encapsulando as regras de negócio que protegem o ativo mais valioso da TurboCat: a confiança de seus clientes.

A stack tecnológica foi selecionada para servir e potencializar esta arquitetura. O **.NET Aspire** nos fornece as fundações para uma aplicação Cloud-Native, implementando nosso Command Bus de forma robusta e performática, e o **RiseOn.ResultRail** garante que o tratamento de sucesso e erro seja explícito e consistente em toda a aplicação.

Com estas fundações estabelecidas, a equipe de desenvolvimento está equipada não apenas com um mapa, mas com uma bússola. Este documento vivo deve ser o guia contínuo para construir uma solução que não apenas organize o "caos funcional", mas que fortaleça e amplifique o diferencial competitivo da TurboCat no mercado.

REFERÊNCIAS

- Evans, Eric. *Domain-Driven Design: Tackling Complexity in the Heart of Software*.
- Vernon, Vaughn. *Implementing Domain-Driven Design*.
- Khononov, Vlad. *Learning Domain-Driven Design*.

GLOSSÁRIO

- **Aggregate Root (Raiz do Agregado)**: Entidade principal de um agregado que controla o acesso e as invariantes de outras entidades e objetos de valor dentro do mesmo agregado.

- **API-First**: Abordagem de design onde a API é o principal produto, e não um subproduto.
- **Bounded Context (Contexto Delimitado)**: Limite explícito dentro de um domínio onde um modelo de domínio específico é aplicável.
- **CQRS (Command Query Responsibility Segregation)**: Padrão que segrega as operações de leitura (Queries) e escrita (Commands) para um armazenamento de dados.
- **Core Domain (Domínio Central)**: A parte mais importante e complexa do negócio, onde a empresa busca sua vantagem competitiva.
- **DDD (Domain-Driven Design)**: Abordagem de desenvolvimento de software que foca na modelagem do domínio de negócio.
- **Event Storming**: Workshop colaborativo para modelar domínios de negócio complexos através da identificação de eventos de domínio.
- **JWT (JSON Web Token)**: Padrão aberto para a criação de tokens de acesso que permitem a autenticação e autorização de forma segura.
- **Linguagem Ubíqua**: Linguagem comum e consistente utilizada por todos os membros da equipe (negócio e desenvolvimento) para descrever o domínio.
- **Mediator**: Padrão de design que reduz o acoplamento entre objetos, fazendo com que eles se comuniquem através de um objeto mediador.
- **Mermaid**: Ferramenta de diagramação baseada em texto que renderiza diagramas e fluxogramas.
- **Message Broker**: Componente que permite a comunicação assíncrona entre diferentes serviços através de mensagens.
- **Monólito Modular**: Arquitetura onde o sistema é desenvolvido como uma única unidade de deployment, mas internamente dividido em módulos bem definidos e desacoplados.
- **MVP (Minimum Viable Product)**: Produto com o conjunto mínimo de funcionalidades para ser lançado e testado no mercado.
- **OHS/PL (Open-Host Service / Published Language)**: Padrão de integração onde um contexto publica uma API bem definida e um modelo de dados estável para consumo por outros contextos.
- **Protobuf (Protocol Buffers)**: Método de serialização de dados estruturados, desenvolvido pelo Google, que é mais eficiente que XML ou JSON.
- **Publish/Subscribe (Pub/Sub)**: Padrão de comunicação assíncrona onde publicadores enviam mensagens para tópicos e assinantes recebem mensagens desses tópicos.
- **RESTful API**: Interface de programação de aplicações que segue os princípios do

estilo arquitetural REST (Representational State Transfer).

- **Vertical Slice Architecture**: Abordagem arquitetural que organiza o código por "features" ou "casos de uso", em vez de por camadas técnicas.
- **VO (Value Object)**: Objeto que representa um conceito do domínio sem identidade própria, definido apenas por seus atributos.
- **Command Bus:** Componente de infraestrutura que recebe comandos e os direciona para um único handler para processamento. Facilita a implementação do padrão CQRS e desacopla a camada de apresentação da lógica de aplicação.
- **Result Pattern:** Padrão de design que encapsula o resultado de uma operação que pode falhar. Em vez de lançar exceções para controle de fluxo, um método retorna um objeto que contém ou o valor de sucesso ou um erro, como implementado pela biblioteca `RiseOn.ResultRail`.
- **Wolverine:** Framework de mensageria e execução de comandos para .NET. Atua como um Command Bus de alta performance, executando handlers de forma in-process ou assíncrona, facilitando a implementação de padrões como Outbox e CQRS.

APÊNDICE A: ARCHITECTURAL DECISION RECORDS (ADRS)

ADR-001: Adoção de Monólito Modular em vez de Microsserviços para o MVP

- **Status:** Aceito
- **Contexto:** Prazo agressivo (1 mês) e equipe enxuta (1 Arquiteto, 1 Dev Sênior).
- **Decisão:** Iniciar com uma arquitetura de **Monólito Modular**. Bounded Contexts serão implementados como projetos .NET distintos dentro da mesma solução, implantados como uma única unidade.
- **Alternativas Consideradas:** Arquitetura de Microsserviços desde o início.
- **Justificativa:** Acelera o desenvolvimento, reduz a sobrecarga operacional para o MVP e mitiga riscos futuros, pois o design modular facilita uma eventual extração para microsserviços.

ANEXOS

Codigos Mermaids:

1. **Mapa de event storming:**

```
graph LR
subgraph "Legenda"
        direction LR
        L_Actor[👤 Ator]
```

```
    L_Cmd(🔵 Comando)
    L_Event(🟠 Evento de Domínio)
    L_Agg{🟡 Agregado/Sistema}
    L_Policy(🟣 Política)
end

subgraph "FASE 1: Recepção e Diagnóstico"
    direction LR
    A1[👤 Cliente] --> C1(🔵 Solicitar Reparo de Veículo)
    C1 --> AGG1{🟡 Ordem de Serviço}
    AGG1 --> E1(🟠 Ordem de Serviço Iniciada)
    E1 --> C2(🔵 Cadastrar Novo Cliente e Veículo)
    C2 --> AGG2{🟡 Cliente}
    C2 --> AGG3{🟡 Veículo}
    AGG2 --> E2(🟠 Cliente Cadastrado)
    AGG3 --> E3(🟠 Veículo Cadastrado)
    E2 & E3 --> C3(🔵 Realizar Checklist de Entrada)
    C3 --> AGG1
    AGG1 --> E4(🟠 Checklist de Entrada Concluído)
    E4 --> C4(🔵 Designar Mecânico para Diagnóstico)
    C4 --> AGG1
    AGG1 --> E5(🟠 Mecânico Designado para Diagnóstico)
    E5 --> C5(🔵 Realizar Diagnóstico Técnico)
    C5 --> AGG1
    AGG1 --> E6(🟠 Diagnóstico Preliminar Concluído)
end

subgraph "FASE 2: Orçamentação e Aprovação"
    direction LR
    E6 --> C6(🔵 Preparar Orçamento)
    C6 --> AGG1
    AGG1 --> E7(🟠 Orçamento Preparado)
    E7 --> C7(🔵 Enviar Orçamento para Aprovação)
    C7 --> AGG1
    AGG1 --> E8(🟠 Orçamento Enviado ao Cliente)
    E8 --> C8(🔵 Aprovar Orçamento)
    C8 --> AGG1
    AGG1 --> E9(🟠 Orçamento Aprovado pelo Cliente)
end
```

```
    subgraph "FASE 3: Execução e Finalização"
        direction LR
        E9 --> P1(🟣 Quando orçamento é aprovado, priorizar OS)
        P1 --> C9(🔵 Priorizar OS para Execução)
        C9 --> AGG4{🟡 Fila de Reparos}
        AGG4 --> E10(🟠 OS Priorizada)
        E10 --> C10(🔵 Iniciar Reparo)
        C10 --> AGG1
        AGG1 --> E11(🟠 Status da OS Alterado para 'Em Execução')

        subgraph "Cenário de Exceção: Falta de Peça"
            E11 --> C11(🔵 Requisitar Peça do Estoque)
            C11 --> AGG5{🟡 Estoque}
            AGG5 --> E12(🟠 Peça em Falta no Estoque Detectada)
            E12 --> P2(🟣 Se peça está em falta, pausar OS e comprar)
            P2 --> C12(🔵 Pausar Ordem de Serviço) & C13(🔵 Solicitar
Compra Urgente)
            C12 --> AGG1
            AGG1 --> E13(🟠 Status da OS Alterado para 'Aguardando Peça')
        end

        subgraph "Cenário Alternativo: Peça do Cliente"
             E11 --> C14(🔵 Fornecer Peça Própria)
             C14 --> AGG1
             AGG1 --> E14(🟠 Peça Externa Adicionada à OS)
        end

        E11 --> C15(🔵 Finalizar Reparo)
        C15 --> AGG1
        AGG1 --> E15(🟠 Reparo Finalizado)
        E15 --> P3(🟣 Quando reparo finaliza, notificar cliente)
        P3 --> C16(🔵 Notificar Cliente para Retirada)
        C16 --> E16(🟠 Cliente Notificado)
        E16 --> C17(🔵 Realizar Pagamento)
        C17 --> AGG1
        AGG1 --> E17(🟠 OS Finalizada)
        E17 --> C18(🔵 Retirar Veículo)
        C18 --> AGG1
        AGG1 --> E18(🟠 Veículo Entregue)
    end
```

```
    %% Definição de Estilos
    style L_Actor fill:#f2f2f2,stroke:#333,stroke-width:2px
    style L_Cmd fill:#99ccff,stroke:#333,stroke-width:1px
    style L_Event fill:#ff9900,stroke:#333,stroke-width:1px
    style L_Agg fill:#ffff99,stroke:#333,stroke-width:2px
    style L_Policy fill:#cc99ff,stroke:#333,stroke-width:1px

    style A1 fill:#f2f2f2,stroke:#333,stroke-width:2px
    style C1 fill:#99ccff,stroke:#333,stroke-width:1px
    style C2 fill:#99ccff,stroke:#333,stroke-width:1px
    style C3 fill:#99ccff,stroke:#333,stroke-width:1px
    style C4 fill:#99ccff,stroke:#333,stroke-width:1px
    style C5 fill:#99ccff,stroke:#333,stroke-width:1px
    style C6 fill:#99ccff,stroke:#333,stroke-width:1px
    style C7 fill:#99ccff,stroke:#333,stroke-width:1px
    style C8 fill:#99ccff,stroke:#333,stroke-width:1px
    style C9 fill:#99ccff,stroke:#333,stroke-width:1px
    style C10 fill:#99ccff,stroke:#333,stroke-width:1px
    style C11 fill:#99ccff,stroke:#333,stroke-width:1px
    style C12 fill:#99ccff,stroke:#333,stroke-width:1px
    style C13 fill:#99ccff,stroke:#333,stroke-width:1px
    style C14 fill:#99ccff,stroke:#333,stroke-width:1px
    style C15 fill:#99ccff,stroke:#333,stroke-width:1px
    style C16 fill:#99ccff,stroke:#333,stroke-width:1px
    style C17 fill:#99ccff,stroke:#333,stroke-width:1px
    style C18 fill:#99ccff,stroke:#333,stroke-width:1px

    style E1 fill:#ff9900,stroke:#333,stroke-width:1px
    style E2 fill:#ff9900,stroke:#333,stroke-width:1px
    style E3 fill:#ff9900,stroke:#333,stroke-width:1px
    style E4 fill:#ff9900,stroke:#333,stroke-width:1px
    style E5 fill:#ff9900,stroke:#333,stroke-width:1px
    style E6 fill:#ff9900,stroke:#333,stroke-width:1px
    style E7 fill:#ff9900,stroke:#333,stroke-width:1px
    style E8 fill:#ff9900,stroke:#333,stroke-width:1px
    style E9 fill:#ff9900,stroke:#333,stroke-width:1px,stroke-dasharray: 5
5
    style E10 fill:#ff9900,stroke:#333,stroke-width:1px
    style E11 fill:#ff9900,stroke:#333,stroke-width:1px
```

```
    style E12
fill:#ff9900,stroke:#333,stroke-width:1px,color:#fff,font-weight:bold
    style E13 fill:#ff9900,stroke:#333,stroke-width:1px
    style E14 fill:#ff9900,stroke:#333,stroke-width:1px
    style E15 fill:#ff9900,stroke:#333,stroke-width:1px,stroke-dasharray:
5 5
    style E16 fill:#ff9900,stroke:#333,stroke-width:1px
    style E17 fill:#ff9900,stroke:#333,stroke-width:1px
    style E18 fill:#ff9900,stroke:#333,stroke-width:1px

    style AGG1 fill:#ffff99,stroke:#333,stroke-width:2px
    style AGG2 fill:#ffff99,stroke:#333,stroke-width:2px
    style AGG3 fill:#ffff99,stroke:#333,stroke-width:2px
    style AGG4 fill:#ffff99,stroke:#333,stroke-width:2px
    style AGG5 fill:#ffff99,stroke:#333,stroke-width:2px

    style P1 fill:#cc99ff,stroke:#333,stroke-width:1px
    style P2 fill:#cc99ff,stroke:#333,stroke-width:1px
    style P3 fill:#cc99ff,stroke:#333,stroke-width:1px
```

2. **Visão de Subdomínios (O Espaço do Problema)**

```
graph TD
    subgraph "TurboCat - Mapa de Subdomínios de Negócio"

        %% Core Domain
        Core("Gerenciamento do Ciclo de Vida do Reparo [Core]")

        %% Supporting Subdomains
        Inventory("Gestão de Inventário e Aquisições [Supporting]")
        Customer("Gestão de Clientes e Veículos [Supporting]")
        Billing("Faturamento e Pagamentos [Supporting]")

        %% Generic Subdomain
        Notifications("Comunicações Transacionais [Generic]")

        %% Relações de Dependência
```

```
        Core -- "precisa de dados de cliente/veículo" --> Customer
        Core -- "precisa de preços e disponibilidade de peças" -->
Inventory
        Billing -- "é acionado por eventos do reparo
(orçamento/conclusão)" --> Core
        Notifications -- "é acionada por mudanças de status no
reparo" --> Core

    end

    %% Estilos
    style Core fill:#ffff99,stroke:#333,stroke-width:2px
    style Inventory fill:#cce5ff,stroke:#333,stroke-width:1px
    style Customer fill:#cce5ff,stroke:#333,stroke-width:1px
    style Billing fill:#cce5ff,stroke:#333,stroke-width:1px
    style Notifications fill:#e6e6e6,stroke:#333,stroke-width:1px
```

3. Visão de Bounded Contexts (O Espaço da Solução)

```
graph TD
    subgraph "TurboCat Core System"
        direction LR

        subgraph "Core Domain"
            FO("<b>FrontOffice Context [D]</b><br>Publica eventos de
negócio (PL)")
            WS("<b>Workshop Context [D]</b><br>Publica eventos de
progresso (PL)")
        end

        subgraph "Supporting Subdomains"
            INV("<b>Inventory Context [U]</b><br>Expõe API de
Inventário (OHS)")
            BIL("<b>Billing Context [D]</b><br>[Pós-MVP]")
        end

        subgraph "Generic Subdomain"
            NOT("<b>Notifications Context [D]</b><br>[Pós-MVP]")
```

```
        end
    end

    subgraph "External Systems"
        EmailSvc("<b>External Email/SMS Service</b><br>(e.g.,
SendGrid)")
    end

    %% Relações de Integração

    %% Relações de Eventos (Pub/Sub)
    WS -- "<b>CF (Conformist)</b><br>Consome 'QuoteApprovedEvent'"
--> FO
    FO -- "<b>CF (Conformist)</b><br>Consome 'RepairCompletedEvent'"
--> WS
    BIL -- "<b>CF (Conformist)</b><br>Consome eventos de
Faturamento<br>(QuoteApproved, RepairCompleted)" --> FO & WS
    NOT -- "<b>CF (Conformist)</b><br>Consome múltiplos eventos de
Status" --> FO & WS

    %% Relações de API (Request/Response)
    FO -- "<b>OHS Client</b><br>Consulta preços de peças" --> INV
    WS -- "<b>OHS Client</b><br>Consulta e requisita peças" --> INV

    %% Relação com Sistema Externo
    NOT -- "<b>ACL (Anti-Corruption Layer)</b><br>Traduz evento para
chamada de API" --> EmailSvc


    %% Estilos
    style FO fill:#E1F5FE,stroke:#01579B,stroke-width:2px
    style WS fill:#E1F5FE,stroke:#01579B,stroke-width:2px
    style INV fill:#FFF9C4,stroke:#F57F17,stroke-width:2px
    style BIL fill:#FFF9C4,stroke:#F57F17,stroke-width:2px
    style NOT fill:#F1F8E9,stroke:#33691E,stroke-width:2px
    style EmailSvc fill:#EEEEEE,stroke:#616161,stroke-width:2px
```

4. Diagrama de Classes dos Agregados

```
classDiagram
    direction LR

    subgraph FrontOffice_Context
        class WorkOrder {
            <<AggregateRoot>>
            +WorkOrderId
            +CustomerId
            +VehicleId
            +Status
        }
        class Customer {
            <<AggregateRoot>>
            +CustomerId
            +Name
        }
        class Vehicle {
            <<Entity>>
            +VehicleId
            +LicensePlate
            +Model
            +Year
        }
        class QuoteLine {
            <<ValueObject>>
            +ItemId
            +Description
            +Quantity
            +UnitPrice
        }
        class CustomerApproval {
            <<ValueObject>>
            +ApprovedBy
            +ApprovalTimestamp
        }
        class ContactInfo {
            <<ValueObject>>
            +Email
            +PhoneNumber
        }
        class DocumentId {
            <<ValueObject>>
            +Number
            +Type
        }

        WorkOrder "1" *-- "1..*" QuoteLine : contains
        WorkOrder "1" *-- "0..1" CustomerApproval : has
        Customer "1" *-- "1" ContactInfo : has
        Customer "1" *-- "1" DocumentId : has
        Customer "1" *-- "0..*" Vehicle : owns
        WorkOrder ..> Customer : refers to
```

```
    WorkOrder ..> Vehicle : refers to
end

subgraph Workshop_Context
    class RepairJob {
        <<AggregateRoot>>
        +RepairJobId
        +WorkOrderId
        +Status
        +AssignedMechanicId
    }
    class Mechanic {
        <<AggregateRoot>>
        +MechanicId
        +Name
        +CurrentAvailability
    }
    class TechnicalNote {
        <<ValueObject>>
        +NoteText
        +AuthorId
        +Timestamp
    }

    RepairJob "1" *-- "0..*" TechnicalNote : contains
    RepairJob ..> Mechanic : assigned to
end

subgraph Inventory_Context
    class InventoryItem {
        <<AggregateRoot>>
        +PartId
        +Name
        +Description
    }
    class Price {
        <<ValueObject>>
        +Amount
        +Currency
    }
    class StockLevel {
        <<ValueObject>>
        +Quantity
        +UnitOfMeasure
    }

    InventoryItem "1" *-- "1" Price : has
    InventoryItem "1" *-- "1" StockLevel : has
end

%% Relações entre contextos
RepairJob ..> WorkOrder : created from
WorkOrder ..> InventoryItem : queries for price
RepairJob ..> InventoryItem : consumes
```

5. **C4 Model: Nível 1: Diagrama de Contexto de Sistema**

```
C4Context
    title System Context for TurboCat Core

    Person(customer, "Cliente", "Dono do veículo que busca o
serviço.")
    Person(admin, "Administrativo/Gerente", "Funcionário da oficina
que gerencia a operação.")

    System(turbocat, "TurboCat Core System", "Gerencia o fluxo de
ordens de serviço, clientes, inventário e comunicação.")

    System_Ext(emailSvc, "Serviço de Email Externo", "Ex: SendGrid,
Mailgun. Responsável pelo envio real das notificações.")

    Rel(customer, turbocat, "Consulta status e aprova orçamentos",
"API (HTTPS/JSON)")
    Rel(admin, turbocat, "Usa o sistema para gerenciar toda a
operação", "API (HTTPS/JSON)")
    Rel(turbocat, emailSvc, "Envia notificações para os clientes",
"API")
```

6. **C4 Model: Nível 2: Diagrama de Contêineres**

```
C4Container
    title Container Diagram for TurboCat Core

    Person(customer, "Cliente")
    Person(admin, "Administrativo/Gerente")

    System_Boundary(turbocat_system, "TurboCat Core (Monólito
Modular)") {
        Container(api, "API Gateway / BFF", "ASP.NET 9", "Ponto de
entrada RESTful. Invoca o Command Bus.")

        Container(frontOffice, "Módulo FrontOffice", ".NET 9
Library", "Gerencia WorkOrders, Clientes e Orçamentos.")
        Container(workshop, "Módulo Workshop", ".NET 9 Library",
"Gerencia Reparos e Mecânicos.")
```

```
        Container(inventory, "Módulo Inventory", ".NET 9 Library",
"Gerencia o estoque de peças.")

        Container(bus, "Message Bus (In-Process)", "Wolverine",
"Desacopla a comunicação interna via Comandos e Eventos.")
        ContainerDb(db, "Banco de Dados", "PostgreSQL", "Armazena o
estado de todos os módulos.")
    }

    Rel(customer, api, "Faz requisições", "HTTPS/JSON")
    Rel(admin, api, "Faz requisições", "HTTPS/JSON")

    Rel(api, bus, "Envia Comandos (ex: CreateWorkOrder)")
    Rel(bus, frontOffice, "Roteia mensagens para os Handlers")
    Rel(bus, workshop, "Roteia mensagens para os Handlers")

    Rel(frontOffice, db, "Lê/Escreve", "EF Core")
    Rel(workshop, db, "Lê/Escreve", "EF Core")
    Rel(inventory, db, "Lê/Escreve", "EF Core")

    Rel(frontOffice, bus, "Publica Eventos (ex: QuoteApproved)")

    Rel(workshop, bus, "Publica Eventos (ex: RepairCompleted)")

    Rel(workshop, frontOffice, "Consome Eventos (via Bus)")

    Rel(frontOffice, inventory, "Consulta dados", "API Interna")
```

7. **C4 Model: Nível 3: Diagrama de Componentes (Foco no Módulo Front Office)**

```
C4Component
    title Component Diagram for Front Office Module (Feature: Create
Work Order)

    Container(api, "API Gateway / BFF")
    ContainerDb(db, "Database", "PostgreSQL")

    System_Boundary(frontOffice_module, "Módulo FrontOffice") {
```

```
        Component(createWorkOrderHandler, "CreateWorkOrder Handler",
"Lógica da Aplicação (Use Case)", "Orquestra a criação da OS. É
invocado pelo Command Bus.")
        Component(workOrderAggregate, "WorkOrder Aggregate", "Modelo
de Domínio", "Encapsula as regras de negócio para a criação da OS.")
        Component(workOrderRepo, "WorkOrder Repository", "Interface
de Persistência", "Define o contrato para salvar a OS.")
        Component(workOrderRepoImpl, "Repository Implementation",
"EF Core", "Implementa a persistência com PostgreSQL.")
    }

    Rel(api, createWorkOrderHandler, "Envia o
'CreateWorkOrderCommand'", "via Command Bus")

    Rel(createWorkOrderHandler, workOrderAggregate, "Usa para
executar a lógica de negócio")
    Rel(createWorkOrderHandler, workOrderRepo, "Usa para solicitar a
persistência")

    Rel(workOrderRepoImpl, workOrderRepo, "Implementa")
    Rel(workOrderRepoImpl, db, "Lê/Escreve em", "SQL")
```