# Documento de Arquitetura de Software – FundCalc

**Versão:** 1.2  
**Data:**

## 1. Introdução

Este documento de arquitetura tem como finalidade descrever a estrutura e as principais decisões arquiteturais do FundCalc, uma aplicação como Serviço (SaaS) para cálculo de fundações. A finalidade é fornecer uma visão clara da arquitetura da aplicação.

## 2. Padrão Arquitetural

O **FundCalc** segue a arquitetura hexagonal (Ports & Adapters. O sistema é dividido em quatro contextos principais – **Negócio**, **Cálculo**, **Análise Técnica/Relatórios** e **Sincronização/Orquestração** – para minimizar acoplamento entre domínios. Cada contexto expõe portas de entrada (APIs HTTP, eventos, webhooks, gRPC) e portas de saída (persistência, mensageria), com separação de comandos e consultas via **CQRS**.

## 3. Objetivos

* **Isolamento do cálculo:** manter o algoritmo de cálculo desacoplado do restante da aplicação e permitir sua execução determinística em *batch* ou no dispositivo local, mesmo em modo offline.
* **Modularização forte por domínio:** evitar acoplamento entre contextos e facilitar a evolução independente de cada módulo.
* **Integridade e auditoria:** garantir versionamento de parâmetros, persistência de eventos e rastreabilidade dos resultados por meio de auditoria e quotas de uso.

## 4. Bounded Contexts

### 4.1 Negócio/Core

Responsável pelo gerenciamento de **usuários**, **licenças** e **assinaturas**, autenticação e autorização, operação offline, cobrança e cancelamento/recuperação. É implementado pelo módulo api\_negocial\_process (Spring Web). As portas de entrada incluem APIs HTTP e *webhooks* de pagamento; as portas de saída compreendem a persistência via **JPA** e mensageria para disparo de eventos. As principais entidades são **Conta de Usuário**, **Licença**, **Assinatura** e **Pagamento**.

### 4.2 Cálculo

Proporciona o cálculo determinístico para **sapatas, estacas, blocos,** é distribuído como api\_batch\_process (módulo *jar*) executado em *batch* e embarcado no aplicativo local. Inclui validações e normalização de unidades. A porta de entrada é o serviço localhostCalculator, invocado pelo aplicativo. As portas de saída englobam a persistência de resultados e a publicação de eventos Resultado Calculado. O motor é isolado do framework para permitir reutilização em diferentes ambientes.

### 4.3 Análise Técnica

Compõe o **parecer técnico** a partir dos dados do Negócio e dos resultados de Cálculo. Aplica regras normativas (como NBRs) e gera o **PDF oficial** do relatório. É implementado pelo módulo api\_tech\_analysis\_process (jar) com portas de entrada para recebimento de dados (GeraParecer) e portas de saída para armazenamento do PDF e emissão de eventos RelatorioGerado. No modo offline, existe uma versão alternativa em memória para uso no aplicativo local.

### 4.4 Sincronização/Orquestração

Implementado pelo módulo api\_sync\_process, esse contexto gerencia a comunicação entre o núcleo e o aplicativo local. Utiliza um **protocolo offline-first** baseado em eventos (*append-only*), validação e reconciliação com semântica por agregados, idempotência, padrões *outbox/inbox* e *checkpoints*. Resolve conflitos de dados entre domínios (por exemplo, parâmetros geotécnicos versus informações de licenças) antes de sincronizar com o banco central. Permite que o aplicativo funcione de forma offline e sincronize posteriormente com o servidor **PostgreSQL**.

## 5. Módulos Principais

1. **Identidade e Autenticação** – gerencia usuários, credenciais, autenticação multifator e tokens de sessão.
2. **Licenciamento e Billing** – controla planos de licenças, métodos de pagamento e cobranças recorrentes. Integra gateways de pagamento e acompanha o ciclo de vida das assinaturas.
3. **Gestão de Projetos e Parâmetros** – administra projetos de fundações, dados geotécnicos, materiais e cargas. Permite a parametrização de unidades e normas de cálculo.
4. **Cálculo** – contém os algoritmos de dimensionamento de sapatas, estacas, s, Fundações etc., com validações e normalização de unidades. É executado em *batch* ou localmente no aplicativo.
5. **Análise Técnica e Relatórios** – compõe e gera relatórios técnicos em PDF com base nos modelos normativos e nos resultados de cálculo.
6. **Sincronização e Resolução de Conflitos** – gerencia o armazenamento de eventos, o estado local versus servidor e as políticas de mesclagem por agregado, garantindo consistência e auditabilidade.

## 6. Fluxo de Dados e Camadas

O aplicativo local utiliza um **banco SQLite** para armazenar dados de usuários, projetos, parâmetros e resultados de cálculo. A camada de sincronização captura eventos gerados nos domínios e envia **deltas** para o servidor central, persistindo-os em **PostgreSQL**. As camadas de serviço api\_local e api\_sync\_process expõem operações para cadastro, cálculo, geração de relatórios e sincronização. A comunicação entre contextos ocorre via eventos e pelas portas definidas (HTTP, gRPC, filas de mensageria).

## 7. Visão de Componentes e Diagramas

A Figura 1 apresenta o **diagrama contextual** de bounded contexts e módulos do FundCalc, destacando as interações entre o aplicativo local, os serviços e o servidor central. Para maior detalhamento, diagramas de componentes e sequência UML podem ser produzidos a partir desse contexto, ilustrando a interação entre usuários, motores de cálculo, camadas de serviço e processos de sincronização.

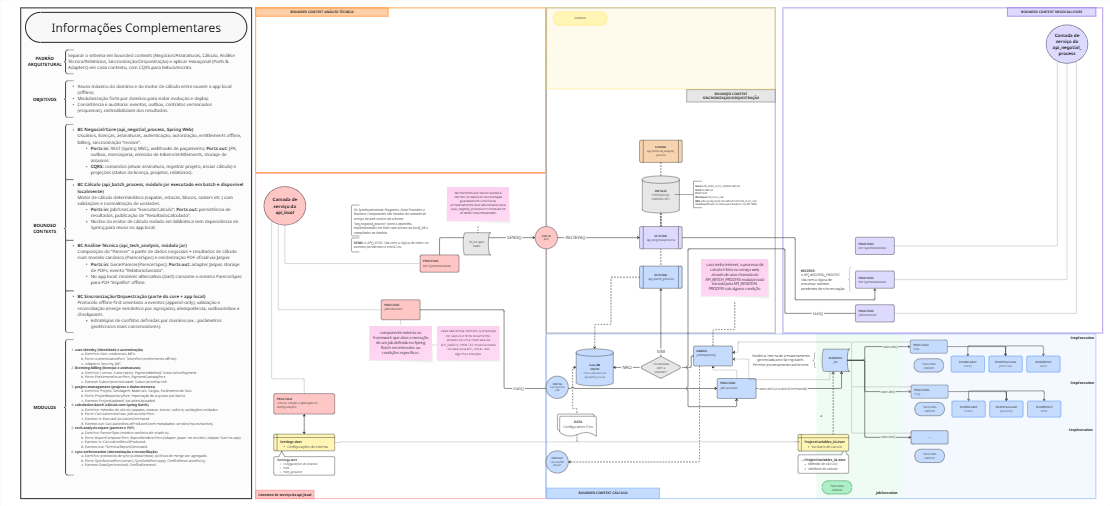


Figura 1 – Diagrama contextual do FundCalc

## 8. Considerações Finais

A arquitetura proposta assegura a separação de responsabilidades, facilita a evolução contínua do sistema e garante que o **FundCalc** opere tanto online quanto offline com consistência e auditabilidade. A utilização de *bounded contexts*, **Ports & Adapters**, **CQRS** e um protocolo de sincronização baseado em eventos cria uma base robusta para ampliar funcionalidades futuras, como integrações com plataformas BIM ou a inclusão de novos métodos de cálculo.