**WattaFood**

**Architecture Notebook**

# Purpose

Este documento descreve a filosofia, decisões, justificativas, elementos significantes e quaisquer aspectos gerais do sistema que formalizam o design e a implementação.

# Architectural goals and philosophy

A maior preocupação da arquitetura será em facilitar a integração de mais plataformas de entrega.

O serviço será acessado por diversas plataformas - a maioria sendo mobile.

Para isso, teremos uma camada de acesso dedicada a prover diferentes métodos de acesso ao sistema

principal.

Não haverão requisitos específicos de hardware.

# Assumptions and dependencies

Dependemos de kits de desenvolvimento (SDKs) das plataformas-alvo. Assumiremos que estas continuarão

saudáveis e disponíveis através dos fornecedores.

# Architecturally significant requirements

Monitoramento da dieta

# Decisions, constraints, and justifications

* Jamais acessar o banco por fora: Haverá uma camada específica para acesso ao banco. Qualquer acesso ao banco deverá ser feito por essa camada
* Classes deverão ter no máximo 700 linhas ou 12 métodos. Se uma classe superar estes limites, deverá ser quebrada em partes menores
* Não usar nenhuma feature das plataformas que seja específico da plataforma. Por exemplo, não usar nenhuma feature de alerta específico das plataformas mobile. A experiência de usuário deverá ser uniforme em todas as plataformas

# Architectural Mechanisms

**Camada de serviço**

Esta é a camada de acesso externo ao serviço principal. Serve para diferenciar plataformas que possam requerer acesso de diferentes métodos. Por exemplo REST ou SOAP.

**Camada de acesso ao banco**

Esta deve ser a única camada permitida de acesso ao banco. Deve conter um mecanismo integrado com o banco de dados de restringir acesso a quaisquer outros componentes.

**Camada de regras de negócio**

Centraliza todas as regras de negócios individuais no mesmo componente.

**Camada de log**

Esta camada irá implementar um sistema de logging para todo o sistema. Usaremos isso para gravar informações sobre a execução do sistema. Estas informações serão usadas para verificar a saúde do serviço e para corrigir problemas no sistema.

**Camada de Integração com Infraestrutura**  
Esta camada se integra com o sistema operacional que executará o processo do serviço de back-end. Precisaremos de serviços do sistema operacional para implementar os alertas que algumas regras de negócio requerem.

# Key abstractions

**Roteador de requisições**

Os pedidos serializados vindos das plataformas-alvo deverão ser despachadas para as regras de negócio correspondentes. Este componente faz este roteamento.

**Formatador de cadastro**

Existem diversos formulários, todos irão ser validados pelo mesmo componente genérico que garantirá a integridade dos dados antes de entrarem no banco.

**Alertas de sistema/plataforma**

Algumas regras de negócio irão disparar alertas (monitoramento por exemplo). Dentro do sistema principal, teremos uma abstração para disparar alertas independente da plataforma.

# Layers or architectural framework

Iremos usar MVC no sistema. O back-end será exclusivamente o Model. O Control será em parte no back-end e em parte na plataforma final (enviando requisitos REST, típicamente). O View será completamente na plataforma final (por exemplo, usando as bibliotecas de UI Android e iOS).

# Architectural views

* **Logical:** Em ordem de fluxo: camada de acesso -> roteador de requisição -> camada de regras de negócio -> camada de banco de dados
* **Operational:** O sistema será single process, mas com uma thread para cada requisição. O banco executará em outro processo.
* **Use case:** Monitoramento da dieta.