ASSOCIAÇÃO EDUCATIVA EVANGÉLICA

Etapa Final

Discentes: Arthur Caixeta de Souza (2310864); Deusmair Junio Souza Prado (2310074);

Ian Correa Couto (2310137)

DOCUMENTAÇÃO
P.I

1. Resumo do Sistema

O Easy Eats é um sistema digital desenvolvido para auxiliar pessoas a organizarem e planejarem suas refeições de forma simples e eficiente. A proposta central é permitir que o usuário informe os ingredientes que possui em casa e, a partir disso, receba sugestões de receitas adequadas, economizando tempo e evitando desperdícios. O público-alvo do sistema são principalmente pessoas com pouca experiência na cozinha, profissionais com rotinas corridas e famílias que desejam manter uma alimentação equilibrada, mas enfrentam dificuldades no planejamento diário. O sistema também se aplica a usuários interessados em nutrição e controle alimentar, permitindo que definam preferências e restrições, como dietas vegetarianas, sem glúten ou com baixo teor calórico. O objetivo geral é tornar o processo de cozinhar mais acessível e personalizado, integrando tecnologia, praticidade e inteligência artificial para recomendar receitas, montar cardápios e gerar listas automáticas de compras.

2. Descrição do Contexto

O Easy Eats será implementado como um aplicativo móvel e uma plataforma web, ambos hospedados em ambiente de nuvem (cloud computing). A arquitetura do sistema foi pensada para suportar um grande volume de acessos simultâneos e permitir evolução contínua sem comprometer o funcionamento dos módulos existentes. O sistema será baseado em uma arquitetura de microserviços, na qual cada funcionalidade principal é desenvolvida como um serviço independente. Essa abordagem favorece o isolamento de falhas, a escalabilidade e a manutenção, além de permitir que diferentes equipes trabalhem em paralelo. O fluxo de uso típico é o seguinte: 1. O usuário cria um perfil e informa suas preferências e restrições alimentares. 2. O Ingredient Service registra os ingredientes disponíveis. 3. O Recipe Service consulta o catálogo de receitas compatíveis com os ingredientes informados. 4. O Recommendation Engine analisa os dados e gera sugestões personalizadas. 5. O Meal Planning Service monta o plano de refeições da semana. 6. O Shopping List Service identifica os itens faltantes e gera uma lista de compras automática. 7. O Notification Service envia lembretes e recomendações diárias. O ambiente técnico será composto por containers Docker orquestrados por Kubernetes, garantindo escalabilidade horizontal, e comunicação entre os serviços feita por meio de REST APIs e mensageria assíncrona (RabbitMQ).

3. Requisitos Funcionais e Não Funcionais

Requisitos Funcionais: - Cadastrar e autenticar usuários com diferentes perfis. - Registrar e gerenciar ingredientes disponíveis. - Buscar receitas baseadas em ingredientes, preferências e restrições alimentares. - Criar planos semanais de refeições. - Gerar listas automáticas de compras com base nas receitas selecionadas. - Enviar notificações e lembretes de preparo ou compra. - Permitir avaliação e feedback das receitas. - Integrar com APIs externas de supermercados e serviços de delivery. Requisitos

Não Funcionais: - Desempenho: Tempo de resposta inferior a 2 segundos nas principais consultas. - Disponibilidade: Uptime mínimo de 99,9% para serviços principais. - Segurança: Autenticação JWT, criptografia de dados e controle de acesso. - Escalabilidade: Possibilidade de adicionar novos serviços sem afetar os demais. - Usabilidade: Interface responsiva e intuitiva em múltiplos dispositivos. - Manutenibilidade: Estrutura modular e versionada, com documentação e testes automatizados. - Compatibilidade: Suporte a navegadores modernos e sistemas Android/iOS.

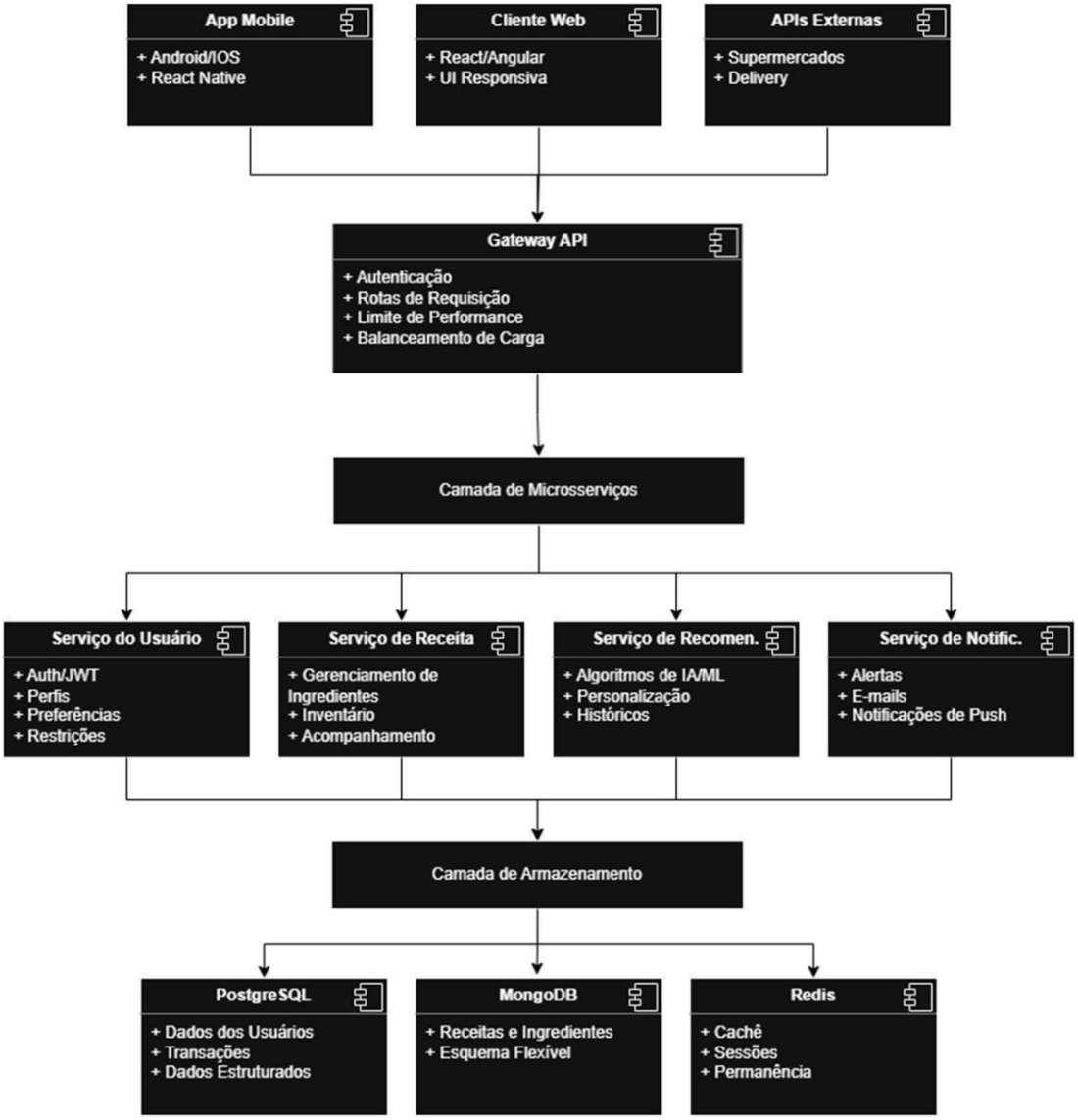
4. Restrições e Premissas Restrições:

A arquitetura do sistema deve obrigatoriamente seguir o padrão de microserviços. - O sistema será hospedado em ambiente cloud com uso de tecnologias open source. - O cronograma de desenvolvimento é curto, exigindo entregas parciais. - O orçamento reduzido limita o uso de serviços pagos de infraestrutura. Premissas: - Usuários terão acesso à internet e dispositivos modernos. - As APIs externas estarão disponíveis e documentadas. - A equipe domina as tecnologias principais (Spring Boot, Docker, Kubernetes e RabbitMQ). - O ambiente de produção oferecerá ferramentas de monitoramento como Prometheus e ELK Stack. 5. Estilo Arquitetural e Justificativa O sistema Easy Eats adotará o estilo arquitetural de microserviços com uso de API Gateway e comunicação assíncrona. Cada serviço é responsável por uma parte específica da aplicação, sendo possível atualizálos e escalar separadamente. Principais componentes da arquitetura: - User Service: Gerencia cadastro, autenticação e preferências do usuário. - Ingredient Service: Armazena e organiza os ingredientes cadastrados. - Recipe Service: Responsável pelo catálogo de receitas e seus metadados. - Recommendation Service: Gera recomendações com base em IA e histórico de uso. - Meal Planning Service: Cria o planejamento semanal de refeições. - Shopping List Service: Gera listas automáticas de compras com base nas receitas. - Notification Service: Envia alertas, lembretes e mensagens personalizadas. Componentes de infraestrutura: - API Gateway: Controla autenticação e roteamento centralizado de requisições. - Service Discovery: Realiza a localização dinâmica dos serviços. - Message Broker (RabbitMQ): Garante comunicação assíncrona e resiliência. - Config Server: Centraliza as configurações dos microserviços. Essa arquitetura foi escolhida porque oferece baixo acoplamento, alta coesão e facilidade de manutenção. Além disso, permite escalabilidade seletiva, onde apenas os módulos mais usados são ampliados, e resiliência, pois falhas em um serviço não interrompem todo o sistema. O uso de diferentes bancos de dados por serviço (PostgreSQL e MongoDB) otimiza o desempenho e respeita o princípio “Database per Service”. O monitoramento será realizado via Prometheus e ELK Stack para garantir rastreabilidade e performance.

5. Diagramas

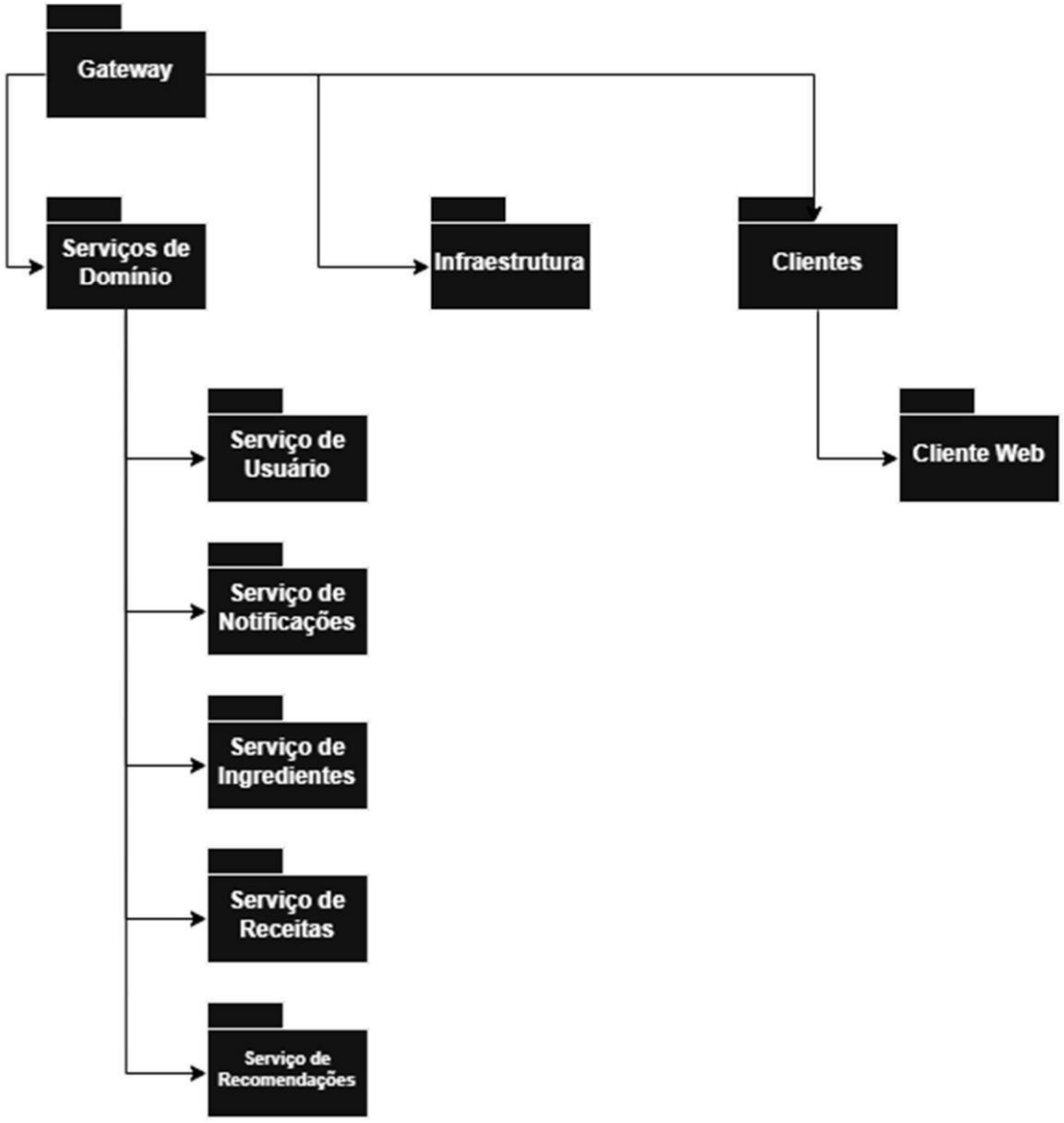
Ferramenta utilizada para os diagramas: Draw.io

5.1 Diagrama de Componentes



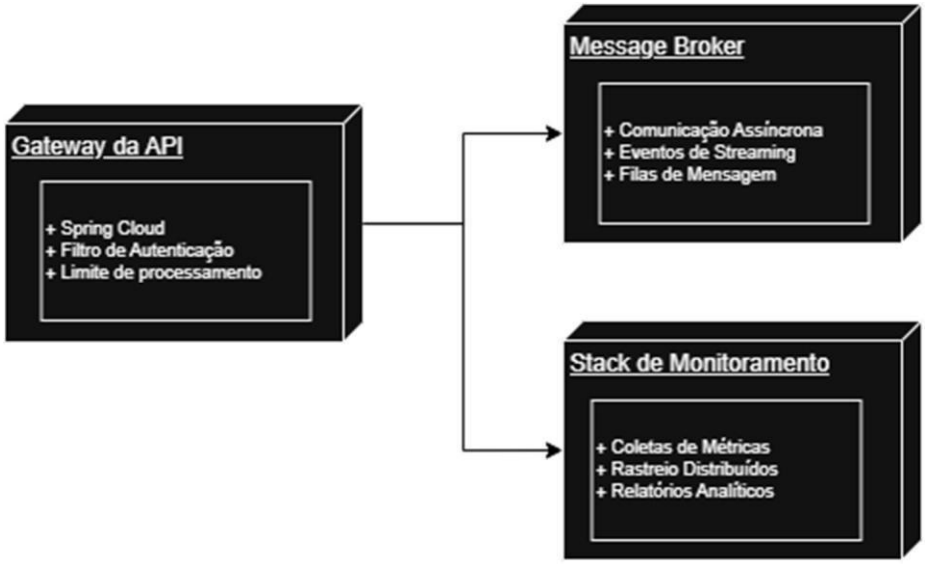
O diagrama apresentado mostra uma arquitetura em camadas que começa com os clientes (aplicativo móvel e web), que se conectam a um API Gateway central. Este gateway distribui as requisições para diversos microserviços especializados - como Usuário, Ingredientes, Receitas e Recomendações - que se comunicam entre si através de um Message Broker e são gerenciados por serviços de infraestrutura como Service Discovery e Config Server. Cada microserviço possui seu próprio banco de dados especializado, enquanto sistemas de monitoramento acompanham todo o funcionamento do sistema, resultando em uma arquitetura desacoplada, escalável e resiliente.

5.2 Diagrama de Pacotes



O diagrama de pacotes organiza o sistema Easy Eats em uma estrutura modular onde cada microserviço é um pacote independente contendo suas próprias camadas de controller, service e repository. Os pacotes são agrupados logicamente em domain-services (lógica de negócio), infrastructure (serviços de apoio), sharedlibraries (código comum) e clients (aplicações front-end), seguindo os princípios de microserviços com baixo acoplamento e alta coesão.

5.3 Diagrama de Implementação



O diagrama de implementação mostra a infraestrutura física do Easy Eats, com todos os componentes executando em um cluster Kubernetes hospedado em nuvem. Cada microserviço é containerizado em pods individuais, gerenciados por serviços de orquestração, enquanto o API Gateway direciona o tráfego externo. A camada de dados utiliza clusters especializados (PostgreSQL para dados transacionais, MongoDB para dados flexíveis, Redis para cache) e o sistema de monitoramento coleta métricas, logs e traços distribuídos para garantir observabilidade total da aplicação.

Gestão de Defeitos e Falhas

Origem

Ao classificar um defeito por origem, estamos querendo dizer a fonte desse defeito, e o que essa fonte pode nos dizer sobre o problema. Para o caso do e-commerce, podemos ter:

- Defeitos de Integração: Essa origem se refere a falhas na comunicação entre os sistemas do aplicativo. Por exemplo, uma API que lista todos os produtos de uma categoria falha, não conseguindo acessar o banco de dados e não retornando os itens pesquisados pelo usuário.

- Defeitos de Interface: Essa origem trata-se de erros visuais que podem atrapalhar a experiência do usuário, mas que não necessariamente envolvem a falha de um processo. Por exemplo, quando o usuário vai para a tela de pagamento após revisar seu pedido, e o layout está completamente bagunçado, com botões e campos fora de lugar e/ou sem formatação.

- Defeitos de Lógica: São erros que envolvem diretamente a lógica do código construído. Por exemplo, o usuário clica em um item do catálogo, o aplicativo o redireciona para uma página aleatória, fora do que foi desejado inicialmente.

- API do Banco de Dados não Retorna Valores: Esse erro poderia ter tanto uma severidade alta, quanto uma prioridade alta também. O usuário não conseguir acessar o catálogo de itens faz com que ele não consiga acessar nenhum produto, podendo gerar uma frustração ou até mesmo a desistência da utilização do sistema. Esse defeito também impediria o usuário de acessar qualquer receita disponível, ou encontrar novas receitas.

- Visual do Aplicativo não Carrega: Esse erro pode ter uma severidade baixa, embora uma prioridade média. O aplicativo não apresentar visuais agradáveis ao usuário não necessariamente o impede de realizar uma compra, porém torna o processo mais trabalhoso e bem menos aceitável, o que também pode acarretar na desistência (ou má impressão) por parte do usuário.

Momento

Quando classificamos um erro pelo momento, estamos querendo dizer em qual fase do desenvolvimento do software ele foi encontrado.

- Durante o Levantamento de Requisitos: um erro pode muito ser encontrado durante o levantamento de algum requisito, principalmente se ele for mal feito. Por exemplo, a equipe de desenvolvimento pode criar uma função que remove itens dos favoritos, sendo que essa função foi “suposta” uma vez que os requisitos não estavam claros.

- Durante uma Revisão Defeitos: são comumente encontrados durante revisões de códigos ou sprints. Por exemplo, um erro que impedia o usuário de acessar determinada página só foi encontrado quando alguém realizou um teste durante a revisão.

- Após o Lançamento: Por fim, erros podem ser encontrados pelos usuários após uma função ter sido lançada, principalmente quando não houve um teste prévio. Por exemplo, uma alteração no sistema de adicionar receitas fez com que usuários não conseguissem visualizar quais receitas tinham, uma vez que as alterações não foram testadas antes do lançamento.

Desenvolvimento em GIT:

https://github.com/IanCoutoFTT/p.i..git

Tela inicial ilustrativa