Documentação do sistema da rede “Seu Cantinho”

Alunos:

Caio Francisco Stadler Sguario - GRR20232340
Eduardo Alencar Furtado Machoski - GRR20232372
João Eduado Zangari Ambrosio - GRR20232344

Github: https://github.com/CaioSguario/design_software

Resumo

Arquitetura

A arquitetura escolhida foi a **orientada a eventos**, um estilo de arquitetura de software baseado na produção, detecção, consumo e reação a eventos. Assim, o sistema será composto por componentes desacoplados que se comunicam por mensagens, denominadas “eventos”.

Para isso, serão definidos produtores de eventos (Publishers), consumidores de eventos (Subscribers) e um Broker de eventos agindo como canal intermediário entre produtores e consumidores.

Essa escolha se deve ao fato de os principais problemas enfrentados pelo antigo sistema estarem relacionados à falta de coerência de dados, intolerância a falhas e falta de escalabilidade. Com essa arquitetura, o sistema consegue utilizar um scheduler para evitar esse tipo de problema, podendo escalar de forma vertical, e utilizando um banco de dados centralizado para evitar erros de sincronia.

Essa arquitetura permite que o sistema escale naturalmente, já que os componentes são desacoplados e se comunicam de forma assíncrona. Ela também deixa o sistema tolerante a falhas, pois o Broker isola a comunicação entre os serviços. Então, caso algum serviço falhe, o resto dos serviços continuam funcionando independentemente.

Dessa forma, oferecendo uma solução robusta a dona Maria, evitando problemas de sincronização entre filiais, assim como inconsistências, o que levavam a uma usabilidade e confiabilidade ruins.

Os principais eventos do sistema são “EspacoCadastrado”, “EspacoAtualizado”, “ReservaCriada”, “ReservaCancelada”, “PagamentoRealizado”, “PagamentoPendente”, e “RelatorioAtualizado”. Esses eventos são essenciais para manter o sistema desacoplado.

No sistema “Seu Cantinho”, o Message Broker serve para que o publicador de eventos não precise conhecer quem irá consumi-lo. Em um projeto real, tecnologias como RabbitMQ ou Apache Kafka poderiam ser usados. Porém, nesse projeto, um broker “fake” foi utilizado, onde apenas as funções de registro e distribuição foram implementadas.

Diagrama de classe

O seguinte diagrama de classes representa a organização de classes do sistema, com seus atributos.

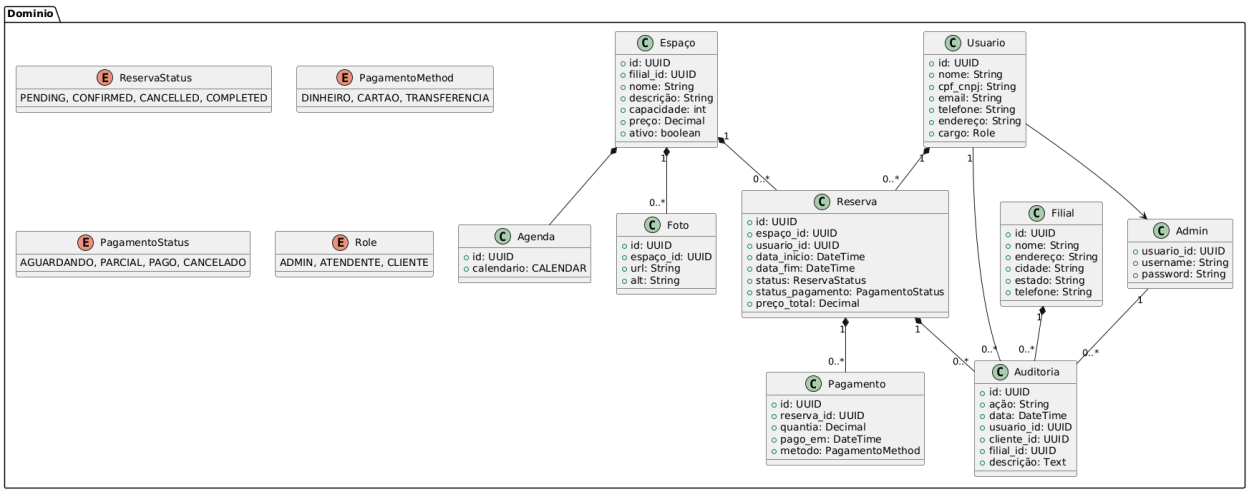
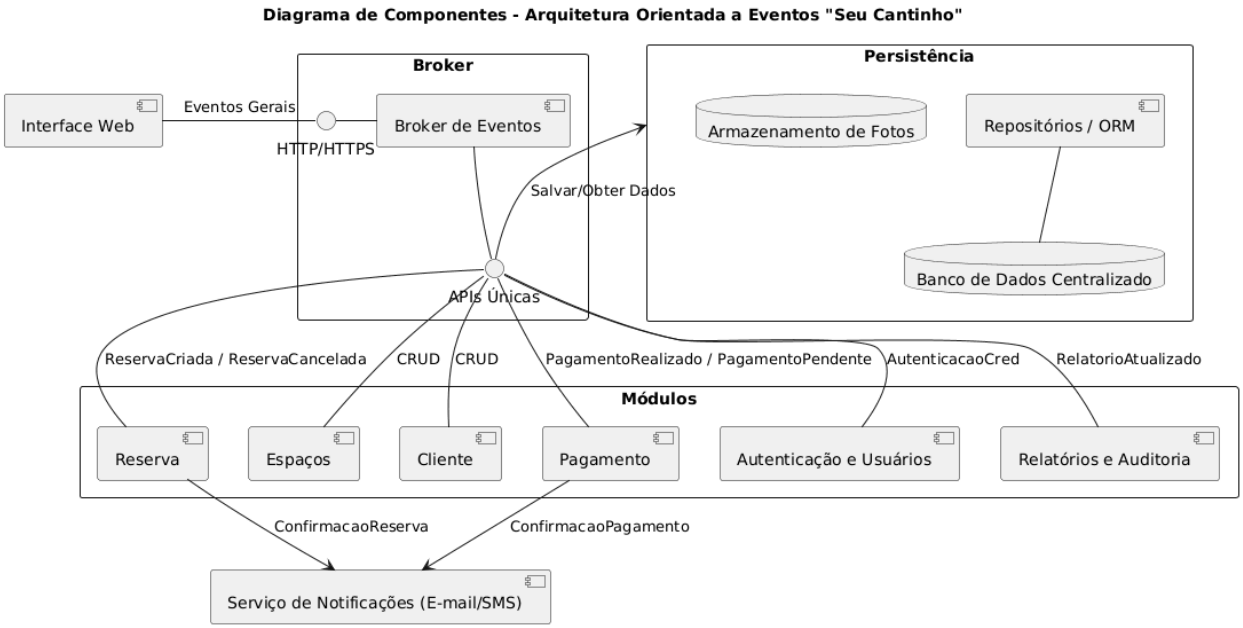


Diagrama de componentes

O seguinte diagrama de componentes representa as relações entre camadas do sistema.



Responsabilidades de cada módulo:

- Interface Web (WebUI)
    - Responsável pela interação direta com o usuário do sistema.
    - Envia e recebe dados através da API REST

- Broker
    - Recebe eventos do front-end e dos módulos.
    - Aciona os módulos necessários para tratamento dos eventos.

    - Intermediário na comunicação entre módulos diferentes.
    - Controle de carga.

- Módulo de Reservas
    - Gerencia as reservas: criação, atualização, cancelamento e confirmação.
    - Evita double-booking

- Módulo de Espaços
    - Gerência cadastros de espaços disponíveis para aluguel
    - Armazena e atualiza informações dos espaços
        - Capacidade
        - Endereço
        - Descrição
        - Fotos
    - Controle de disponibilidade de cada espaço

- Módulo de Clientes
    - Gerencia os clientes: Registro, atualização

- Módulo de Pagamentos
    - Processamento dos pagamentos das reservas

- Módulo de Autenticação
    - Garante a segurança do acesso ao sistema
    - Controle de login, cadastro e permissões de usuários

- Módulo de Relatórios e Auditoria
    - Reúne e organiza dados do sistema para geração de relatorios
    - Registro de ações administrativas e eventos do sistema

- Repositórios/ORM
    - Intermediação entre os módulos de negócio e o banco de dados
    - Abstração para consultas e transações seguras

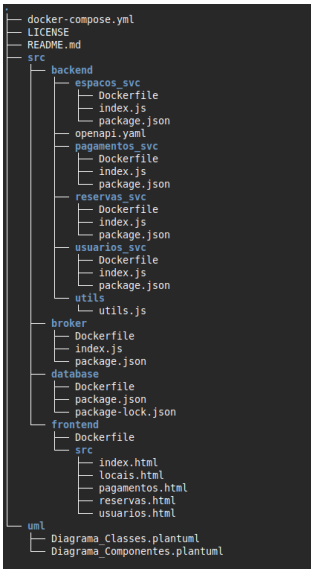
- Banco de Dados Centralizado
    - Armazena dados relacionais principais
        - Clientes
        - Reservas
        - Pagamentos
        - Usuários
        - Espaços

Protótipo:

O protótipo foi realizado utilizando nodejs para o seu backend e comunicação via REST API, para o front-end foi utilizado HTML5, enquanto o back-end foi implementado em arquivo json.

Organização:

O código foi organizado conforme a árvore de diretórios apresentada na imagem, onde cada módulo possui sua pasta e código próprio, e é executado em um container docker independente, se comunicando apenas via REST API com os os outros.



ATULIZAR IMAGEM ANTES DE ENVIAR