Universidade Federal de Goiás
Instituto de Informática
Arquitetura de Software

Documentação Arquitetural com ISO 42010

Alunos: Gustavo Henrique de Freitas Martins
Kaio Ribeiro Sanchez
Gustavo Ribeiro de Oliveira
Matheus Vinicius Dias Ribeiro

Fidelícias

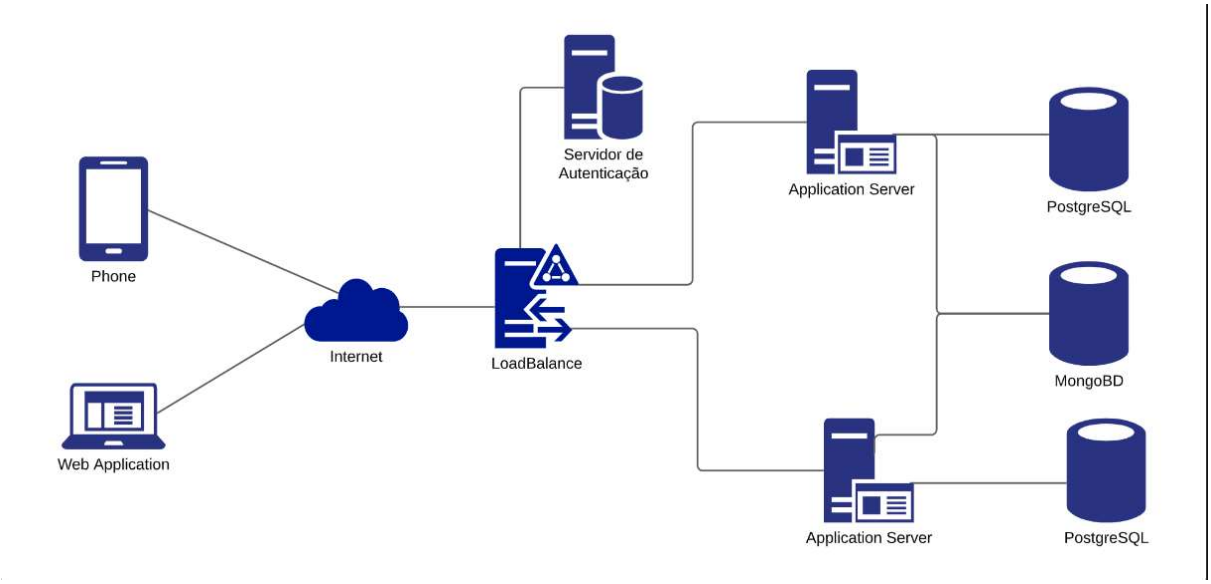
GOIÂNIA
2021

1. Introdução

A ISO 42010 define requisitos sobre a descrição do sistema, software e arquiteturas corporativas. Tem como objetivo padronizar a prática da descrição de arquitetura , definindo termos padrão, apresentando uma base conceitual para expressar, comunicar e revisar arquiteturas e especificar requisitos que se aplicam a descrições de arquitetura , frameworks de arquitetura e linguagens de descrição de arquitetura.

2. Descrição da arquitetura

O sistema Fidelícias será desenvolvido através da utilização da arquitetura de Cliente-Servidor, retratada no Design Arquitetural abaixo:



A arquitetura cliente servidor é uma arquitetura de aplicação distribuída, ou seja, na rede existem os fornecedores de recursos ou serviços a rede, que são chamados de servidores, e existem os requerentes dos recursos ou serviços, denominados clientes.
O cliente não compartilha nenhum de seus recursos com o servidor, mas no entanto ele solicita alguma função do servidor, sendo ele, o cliente, responsável por iniciar a comunicação com o servidor, enquanto o mesmo aguarda requisições de entrada.

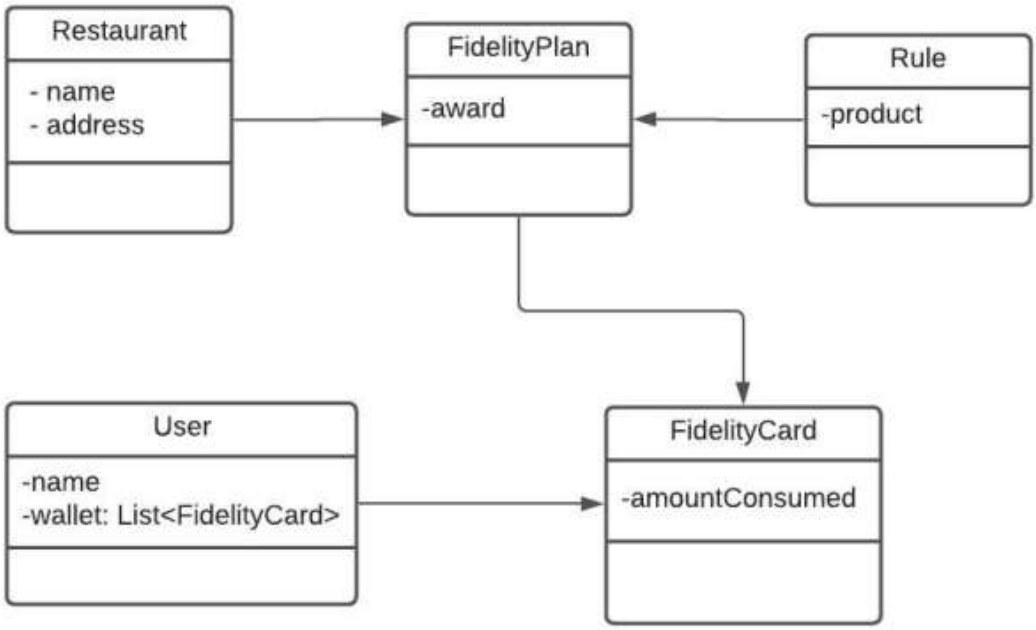
3. Visões arquiteturais e ponto de vista da arquitetura

As visões arquiteturais são os possíveis modos como as pessoas que desempenham papéis diferentes dentro do processo de desenvolvimento de software vêem o problema, tendo diversas perspectivas a fim de melhor identificar o contexto a ser desenvolvido. Ademais, cada visão é registrada utilizando um

determinado modelo, como o desenvolvimento, lógico, físico, processo entre outros. No Fidelicia será abordado o modelo de visão arquitetural lógico.

3.1 Visão lógica

Uma visão lógica mostra as abstrações fundamentais do sistema como objetos ou classes. Nesse tipo de visão, deve ser possível relacionar os requisitos do sistema com as suas entidades.



- Restaurante (Restaurant)
Responsável por cadastrar e manter os dados de um restaurante que utilizará do sistema para o gerenciamento de seus cartões de fidelidade. Um restaurante pode definir vários planos de fidelidades e suas regras desde que sejam diferentes.

- Plano de fidelidade (FidelityPlan)
É responsável por armazenar as informações referente a relação entre Restaurante e Cartão de Fidelidade, gerindo as regras que permitem ao usuário obter os benefícios do plano de fidelidade. Essa classe possui a Regra e o prêmio que o usuário pode receber ao cumprir os requisitos estabelecidos na regra. Um restaurante pode estipular mais de um plano de fidelidade com regras diferentes.

- Regra (Rule)
Responsável por definir quais os requisitos que o plano de fidelidade irá aplicar para que um usuário seja beneficiado com a promoção.

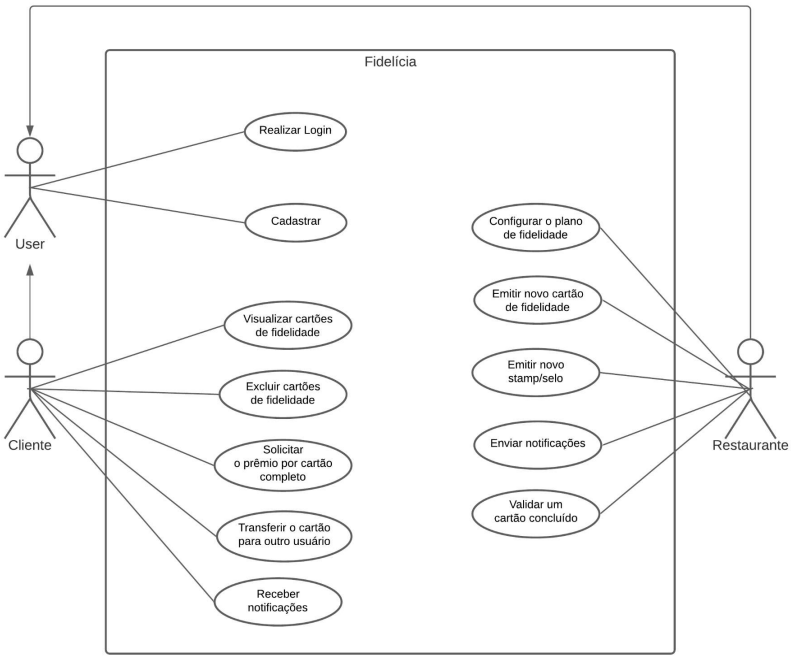
- Cartão de fidelidade (FidelityCard)
Este objeto possui a função de registrar a evolução do usuário no cumprimento das regras de um plano de fidelidade. Ao ser completado todos os requisitos da regra, o cartão torna-se elegível para solicitação do prêmio (award) vinculado ao plano de fidelidade.

- Usuário (User)
O usuário é responsável por realizar um cadastro na aplicação e manter dados de seus “cartões” de fidelidade que estão cadastrados, tendo como objetivo juntar selos para receber promoções ou benefícios. O usuário possui uma lista de cartões de fidelidade aqui chamada de carteira (wallet). Ele pode possuir cartões fidelidade de vários restaurantes ou vários de um mesmo restaurante desde que possuam planos de fidelidade diferentes. 2.Design Arquitetural.

3.2 Visão de cenários

Através da visão de cenários apresentamos um ponto de vista mais próxima do usuário, descrevendo cenários de uso da aplicação. Desta forma apresentaremos essa visão apresentando o diagrama de casos de uso e o digrama de sequencia.

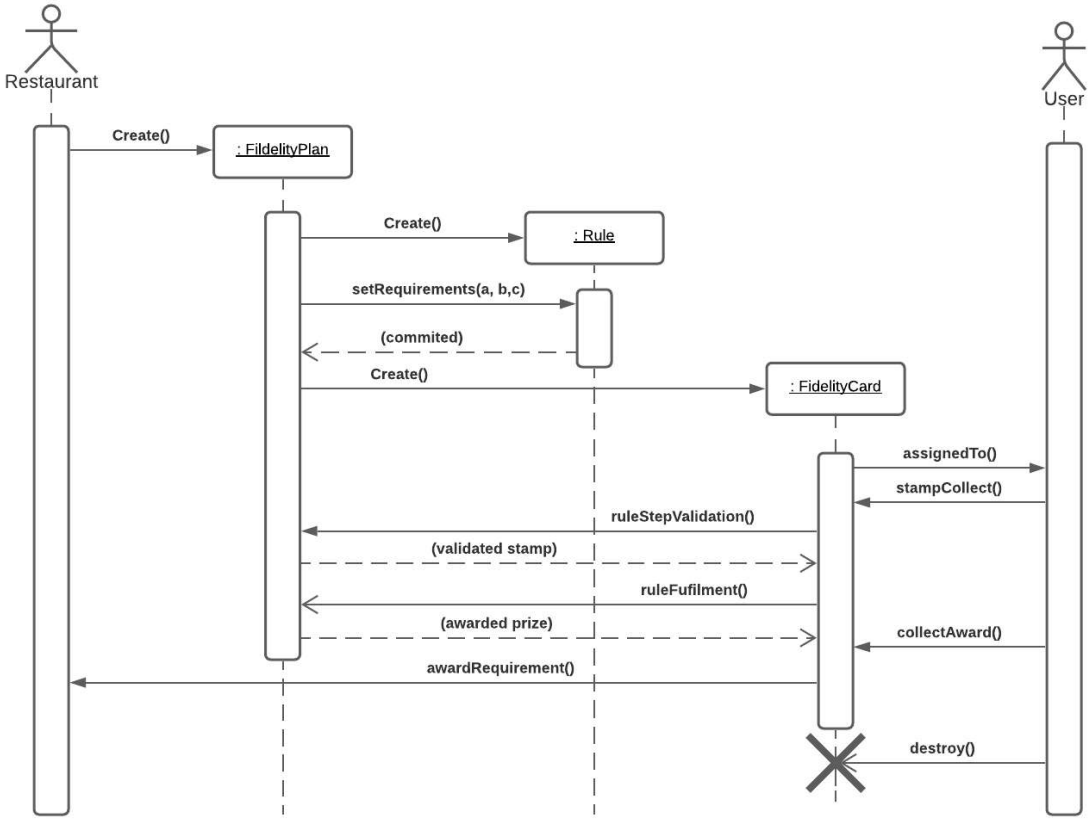
Diagrama de Casos de Uso:



Neste diagrama notamos os dois papeis presentes no sistema que são o cliente e o restaurante. Ambos podem se cadastrar e realizar login no

sistema. Porém após autenticado, cada papel tem as suas próprias atribuições e acessam as funcionalidades que lhe interessam.

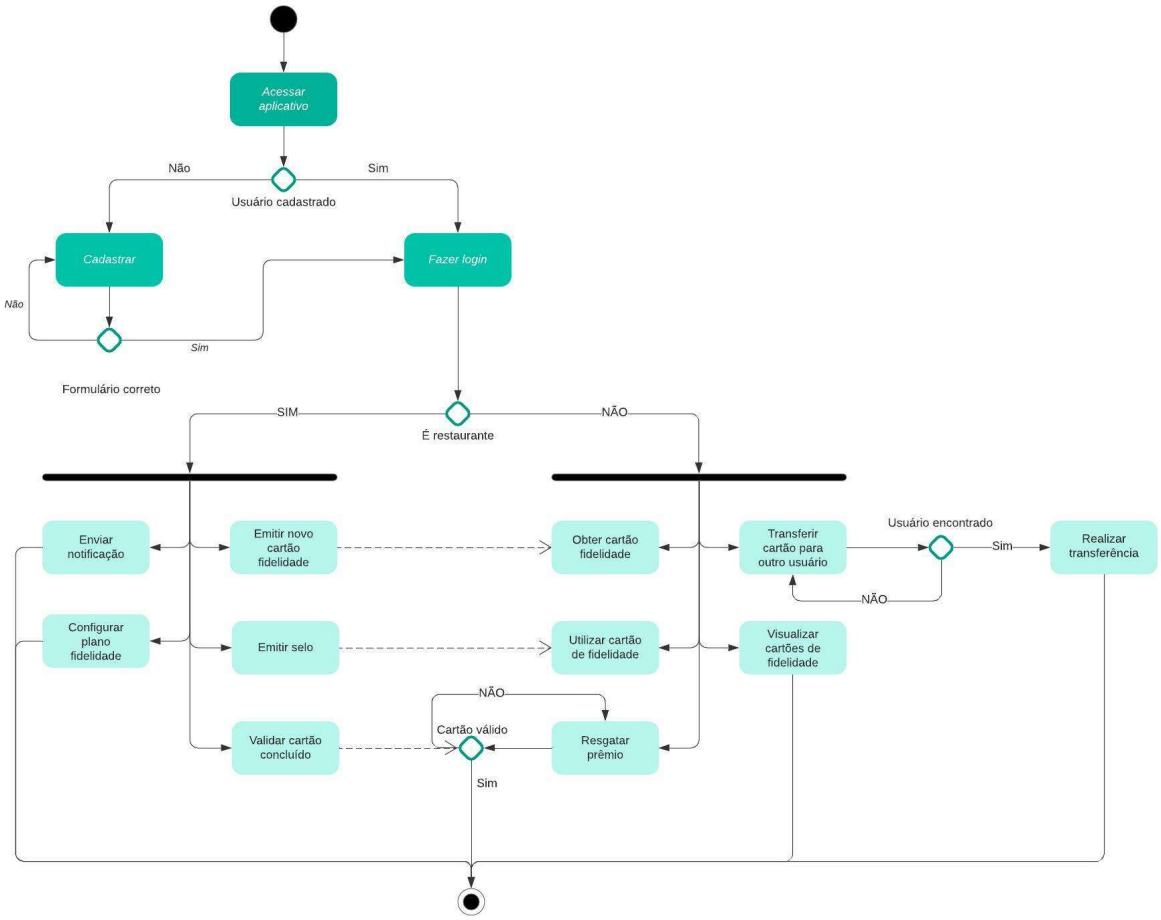
-Diagrama de Sequência:



O diagrama de sequência dá ênfase à ordenação temporal das mensagens. Conforme podemos visualizar no diagrama exposto acima, temos a ordem que as mensagens transitam neste diagrama, Observamos os dois papeis principais representados por seus específicos objetos e como eles influenciam na criação dos demais objetos e como estes são chamados a cumprirem as suas funções.

3.3 Visão de negócio

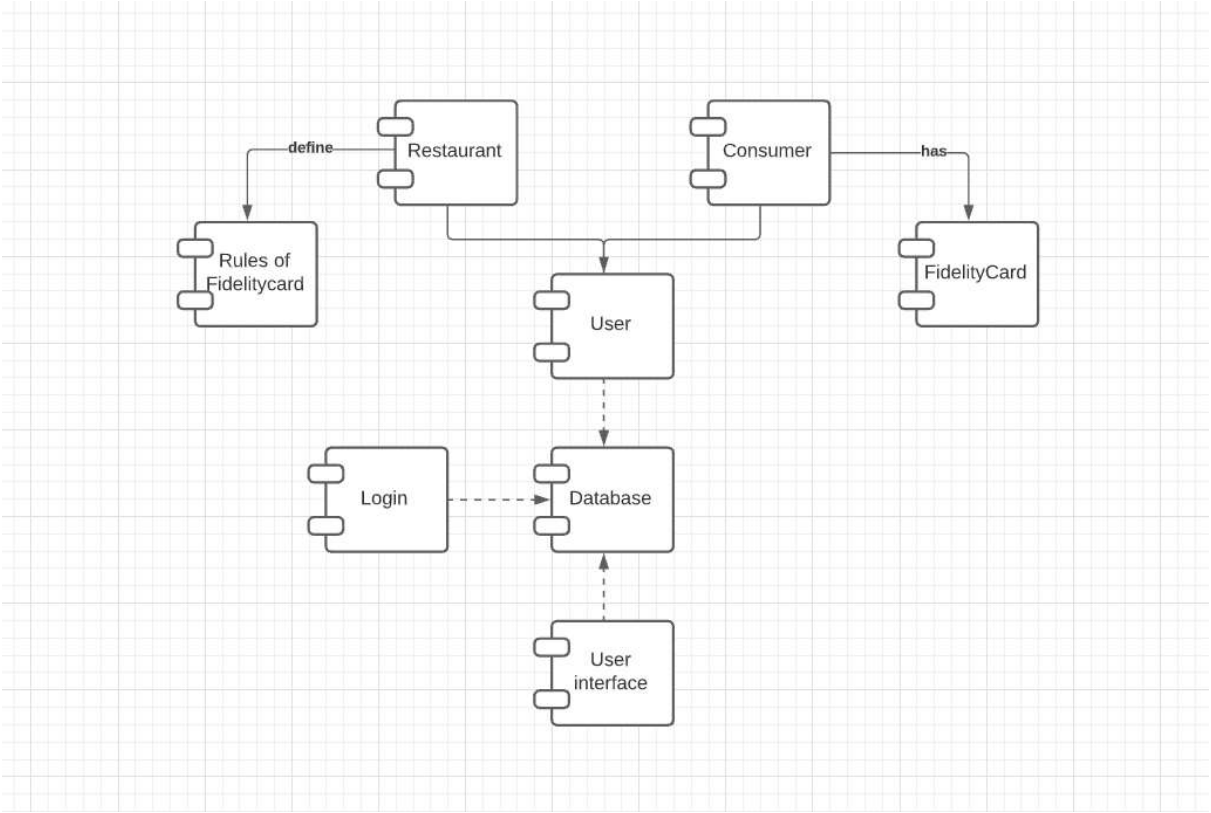
Diagramas de atividade são um tipo de diagrama UML. Os diagramas são usados para mostrar as funcionalidades de várias atividades e fluxos em processos de negócios e sistemas de software.



3.4 Visão de Implementação

A visão de implementação ilustra o sistema do ponto de vista do programador e se preocupa com o gerenciamento de projeto. O sistema Fidelícias será implementado através da utilização dos conceitos de programação orientada a objetos. A linguagem de programação utilizada será React como Front-end e PHP como back-end. Um sistema relacional e um não relacional (NoSQL) serão utilizados como banco de dados.

O diagrama de componentes identifica os componentes que fazer parte de um sistema, um subsistema, e classes de um componente interno. Segue abaixo uma representação do diagrama de pacotes relacionado aos componentes do projeto:



4. Justificativas

No que diz respeito à Arquitetura de Software, sempre é necessário uma tomada de decisões antes, durante e após o desenvolvimento do sistema. A arquitetura tem como principal vantagem a definição e monitoramento do processo de produção desse software. Através dela é possível realizar as decisões arquiteturais, às quais são compostas de descrições, objetivos e fundamentações.
No Fidelícias, escolhemos a arquitetura Cliente-Servidor, pois assim como mencionado e justificado anteriormente, esta contempla aos requisitos do software em questão a ser desenvolvido.