Documento de Arquitetura de Software

1. Introdução

Este documento apresenta a especificação de requisitos contemplados pelo sistema Money Tracker.

1.1 Finalidade

Este documento oferece uma visão geral arquitetural abrangente do sistema, usando diversas visões arquiteturais para representar diferentes aspectos do sistema. O objetivo deste documento é capturar e comunicar as decisões arquiteturais significativas que foram tomadas em relação ao sistema.

1.2 Escopo

O Documento descreve casos de uso de um sistema de controle de finanças o Money Tracker que permite o usuário realizar cadastro de contas bancárias, dinheiro, créditos, depósitos a receber e ativos. E visualizar o balanço e relatórios ao longo do tempo..

Os requisitos especificados neste documento estão relacionados com os casos de uso contidos no documento de especificação de casos de uso.

1.3 Definições, Acrônimos e Abreviações

CSU - Casos de Usos

1.4 Referências

1.4.1 Casos de Uso Contas(CSU_Contas);
1.4.2 Casos de Uso Conversão(CSU_Conversão);
1.4.3 Casos de Uso Painel(CSU_Painel);
1.4.4 Casos de Uso Relatórios(CSU_Relatórios).
1.4.5 Casos de Uso Transações(CSU_Transações).

1.5 Visão Geral

O sistema Money Tracker é um sistema para controle de finanças onde o usuário pode realizar o cadastro de contas contas bancárias, dinheiro, créditos, depósitos a receber e ativos. E o sistema permite o usuário a visualizar facilmente relatórios e balanços sobre as suas finanças.

2. Representação Arquitetural

A arquitetura do sistema é aplicação monolítica, ela descreve uma única aplicação de software em camadas no qual a interface de usuário e código de acesso aos dados são combinados em um único programa a partir de uma única plataforma. Uma aplicação monolítica é autônoma e independente de outras aplicações de computação. A filosofia do projeto consiste em um aplicativo que não é responsável apenas por uma determinada tarefa, mas que também pode executar todos os passos necessários para completar uma determinada função.

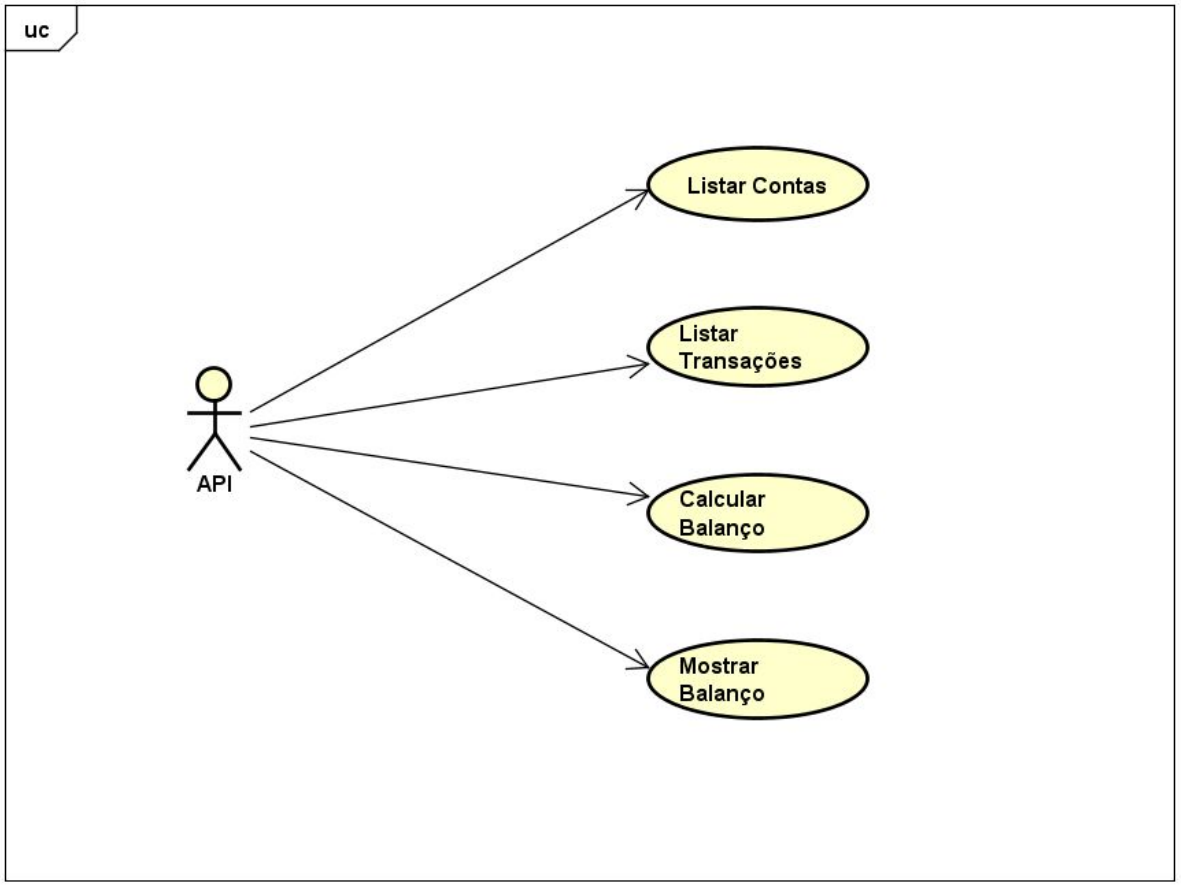
Um aplicativo monolítico descreve uma aplicação de software que é projetado sem modularidade. A modularidade é desejável, em geral, uma vez que suporta a reutilização de partes da lógica da aplicação e também facilita a manutenção, permitindo o reparo ou substituição de peças da aplicação sem a necessidade de substituição por atacado. A modularidade é obtida em graus diversos, por diferentes abordagens de modularização. Código baseado em modularidade permite aos desenvolvedores reutilizar e reparar as partes do aplicativo, mas ferramentas de desenvolvimento são necessários para executar essas funções de manutenção (por exemplo, a aplicação pode precisar ser recompilada). Objeto baseado em modularidade prevê a aplicação como uma coleção de separar os arquivos executáveis que podem ser mantidos de forma independente e substituído sem reimplantar o aplicativo inteiro (por exemplo, arquivos "dll" da Microsoft, arquivos "shared object" da Sun/UNIX). Alguns recursos objeto de mensagens permite aplicações baseadas em objeto

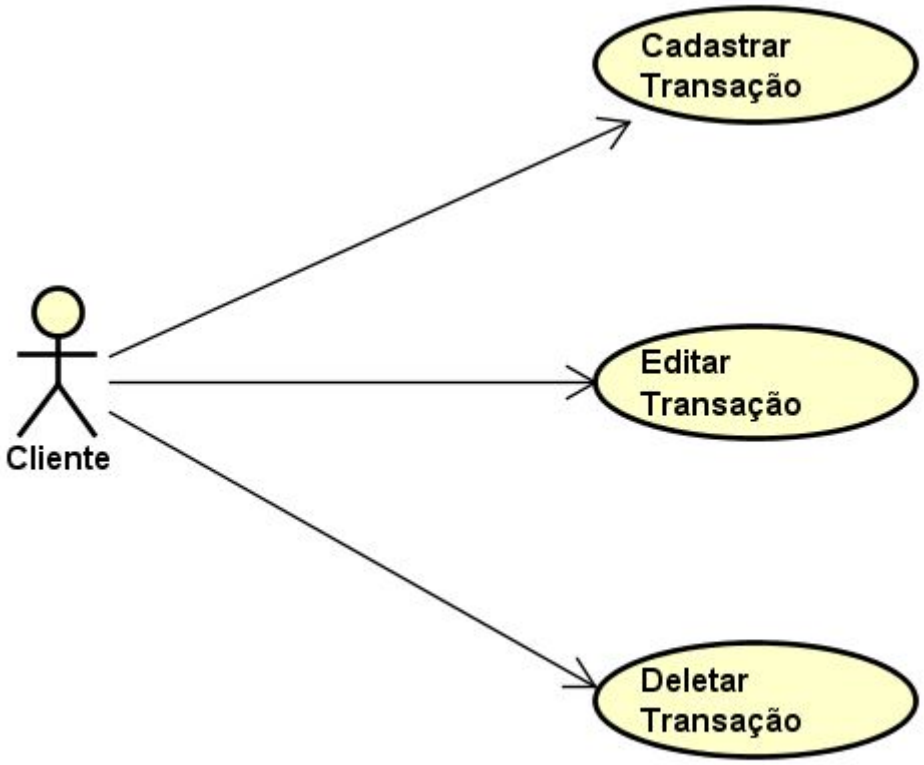
serem distribuídos em vários computadores (por exemplo, Microsoft COM +). Service Oriented Architectures uso padrão de comunicação específica e protocolos de comunicação entre os módulos

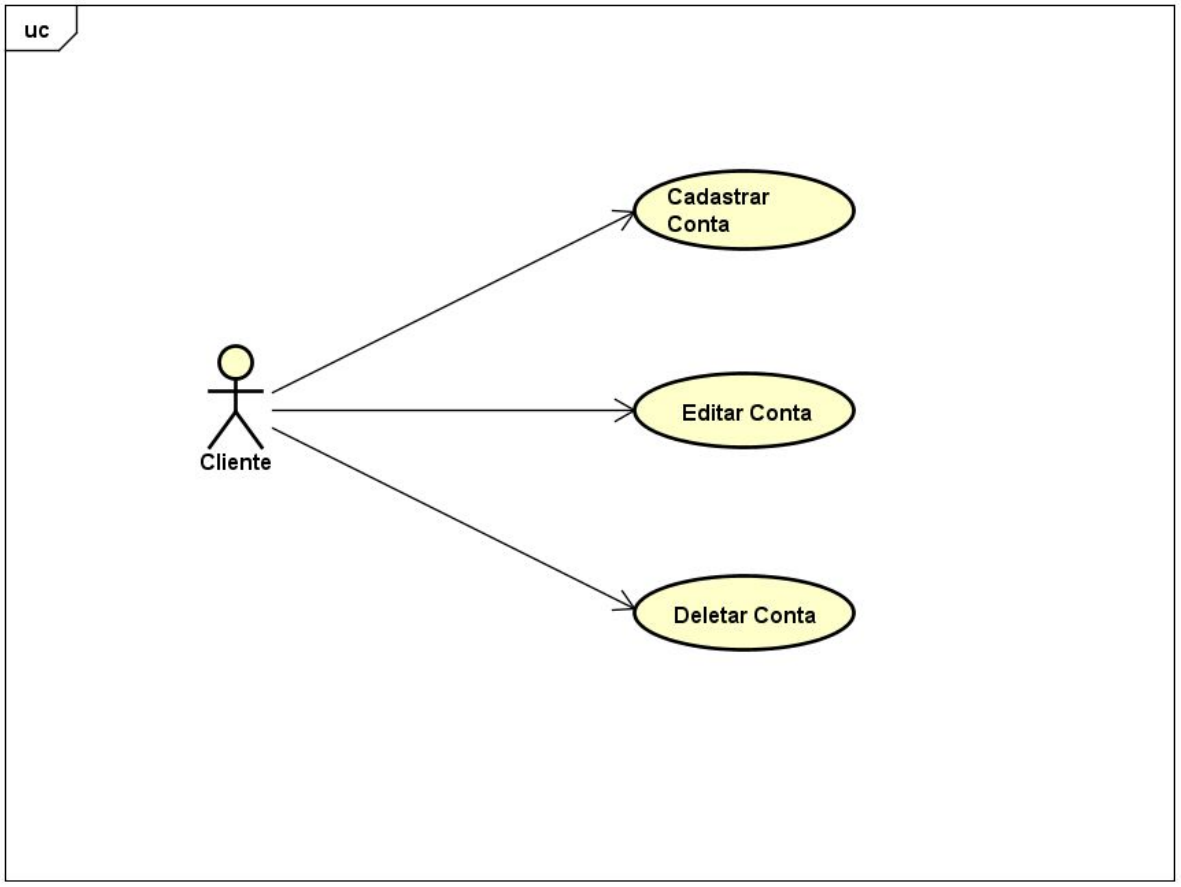
3. Metas e Restrições da Arquitetura

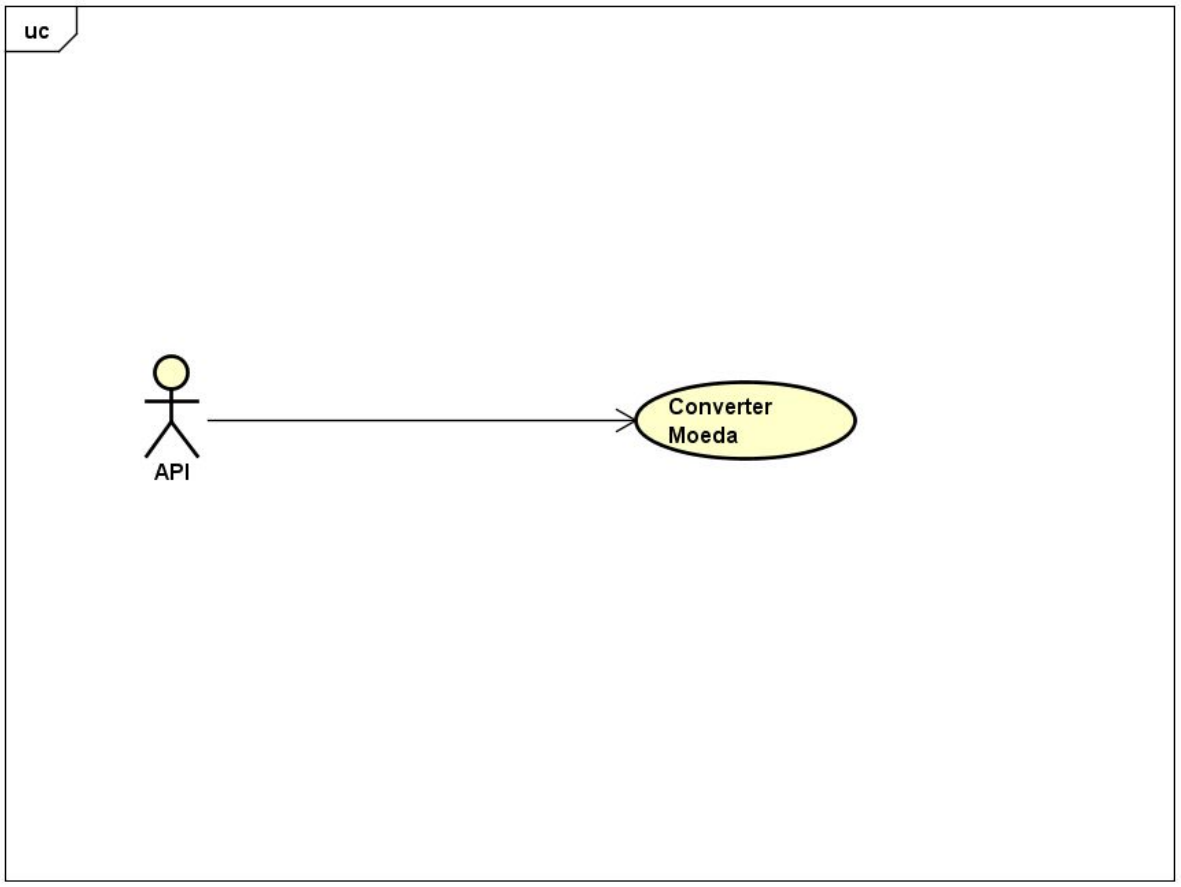
- Todos os requisitos de desempenho e carga, conforme estipulado no Documento de Visão e da deve ser levado em consideração, como a arquitetura está sendo desenvolvido.
- O Sistema exibirá as informações das contas do usuário mesmo sem acesso à internet.
- O Sistema só permitirá atualização das contas se houver acesso à internet.
- O Sistema deverá contar criptografar todos os dados dos usuários e garantir a segurança do mesmo.
- O Sistema deverá ser disponível para iOS, Android e WEB.
- O Sistema deverá contar com uma interface simples e de fácil entendimento.
- O Sistema precisará estar ativo sempre.

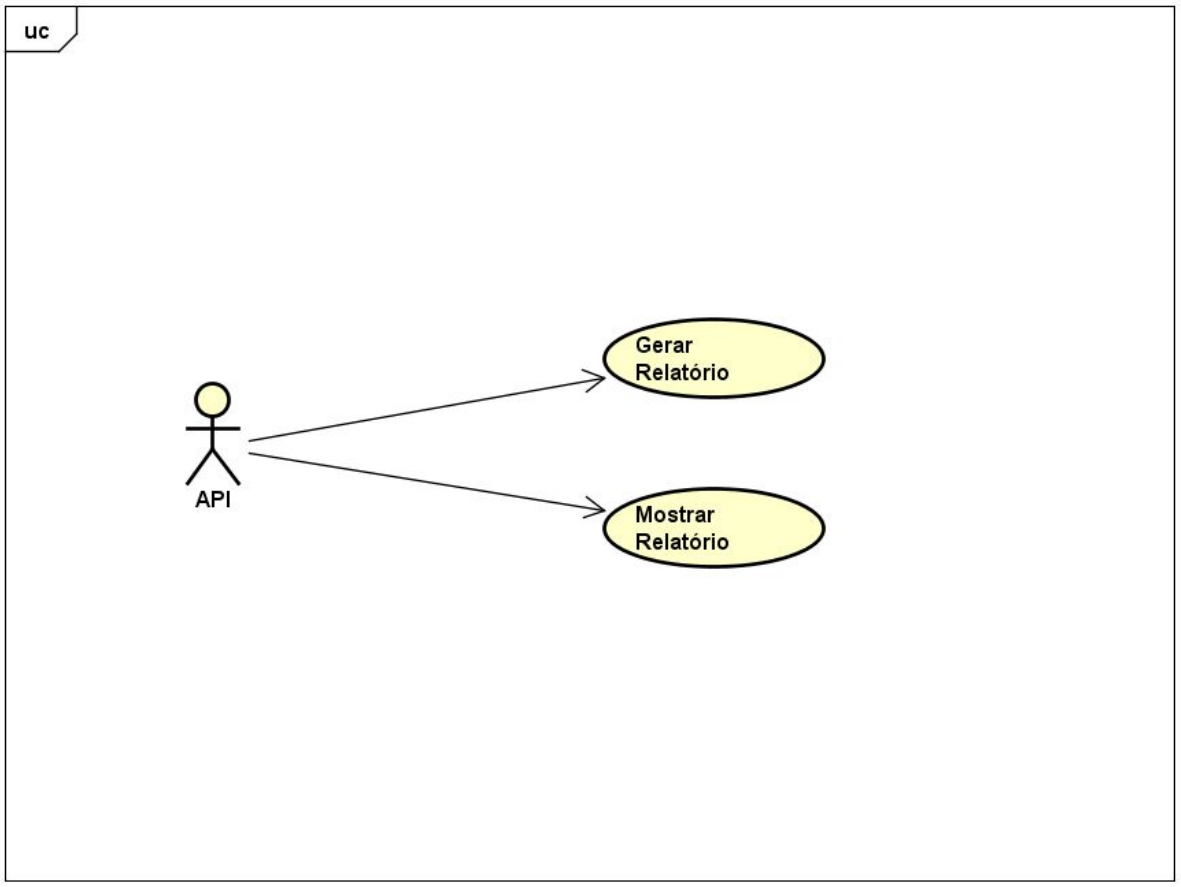
4. Visão de Casos de Uso



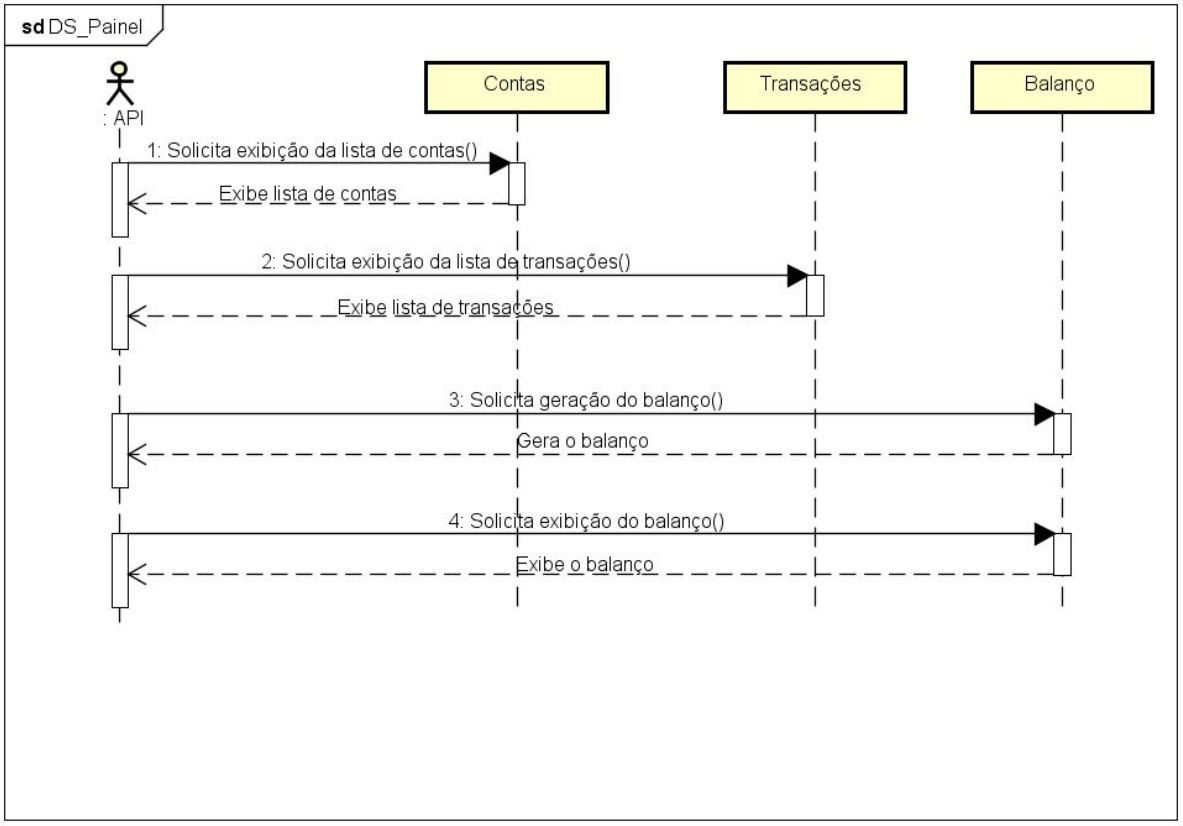


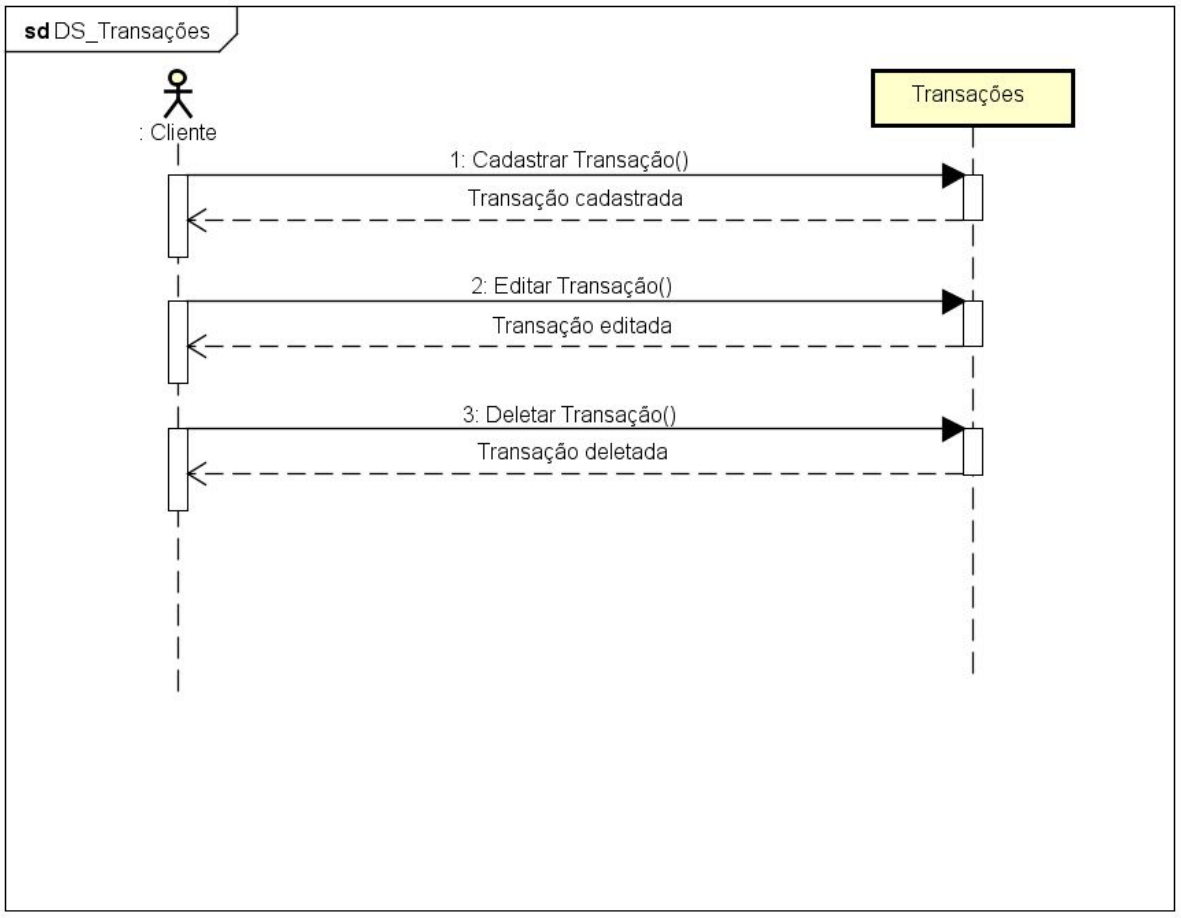


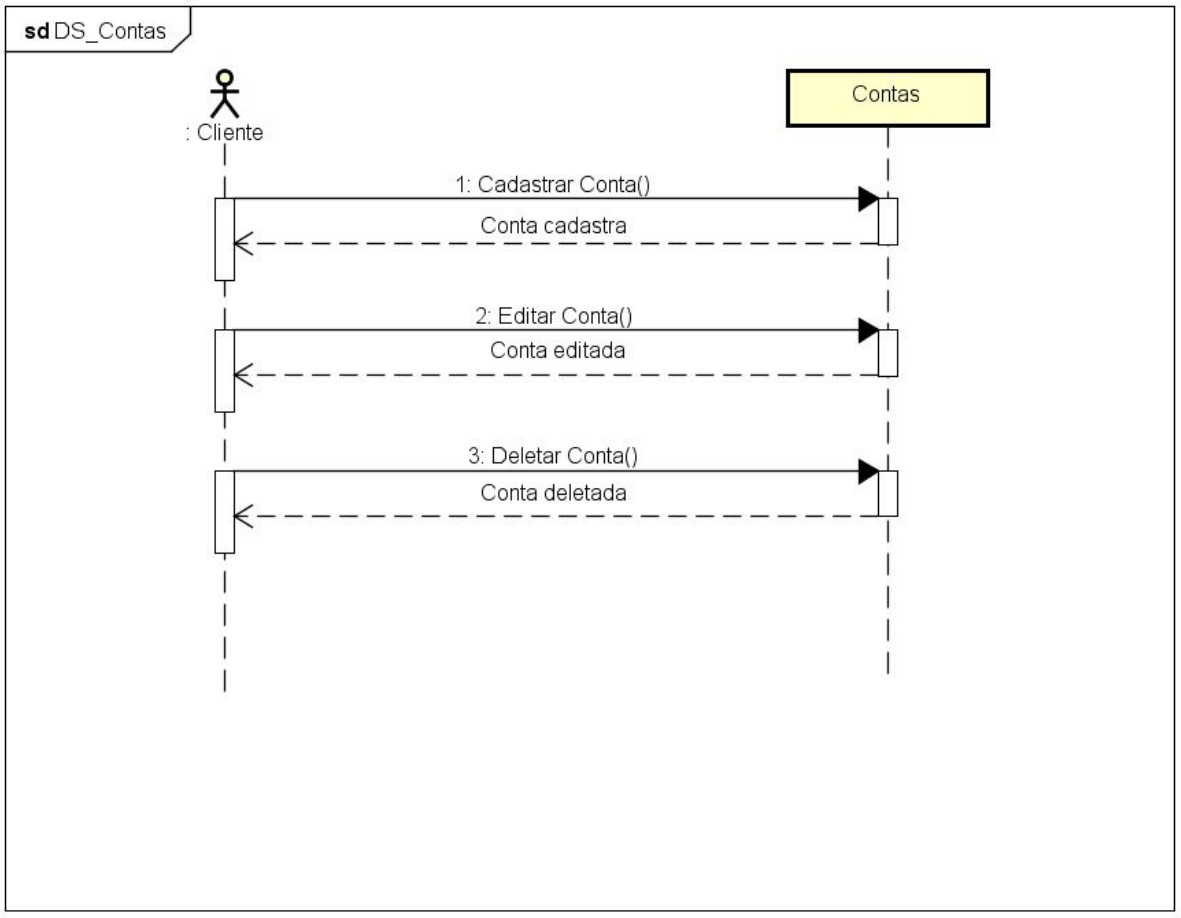


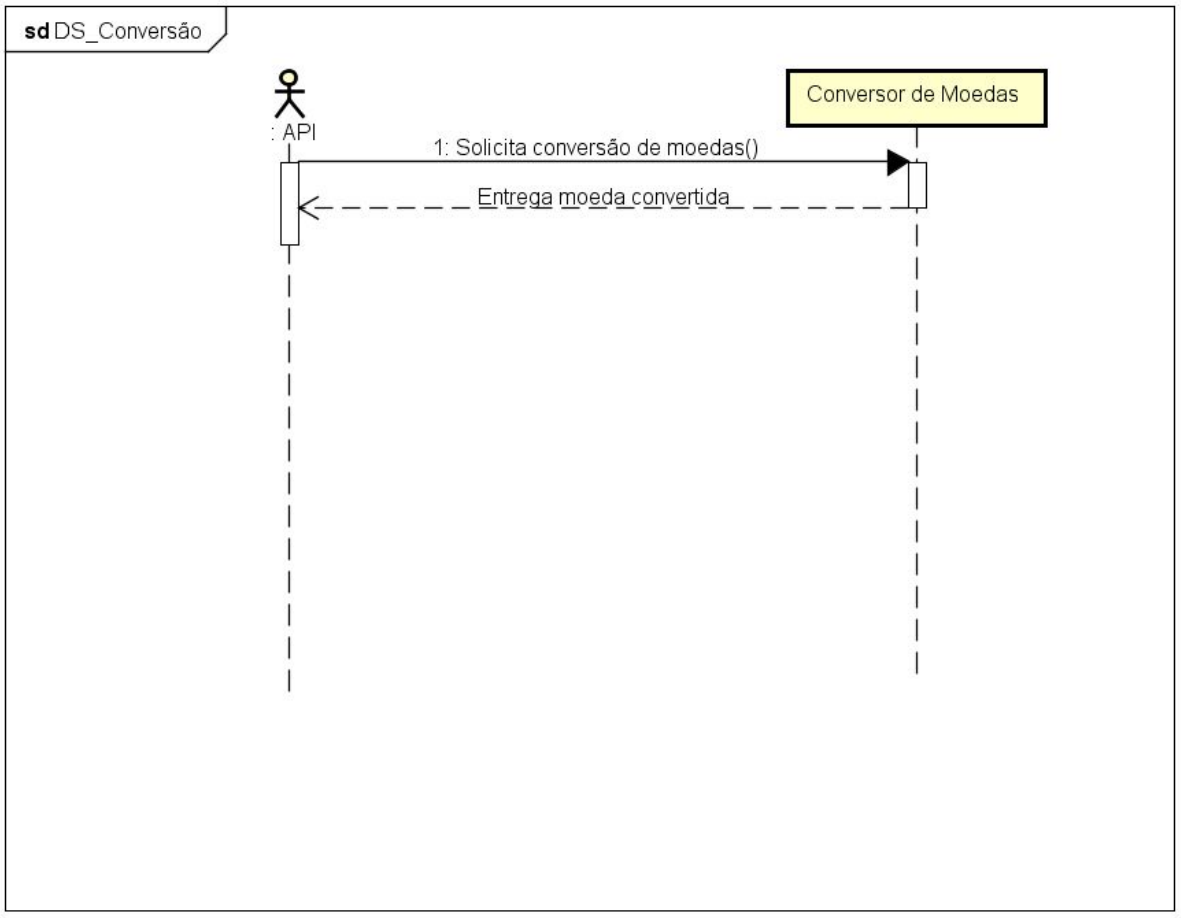


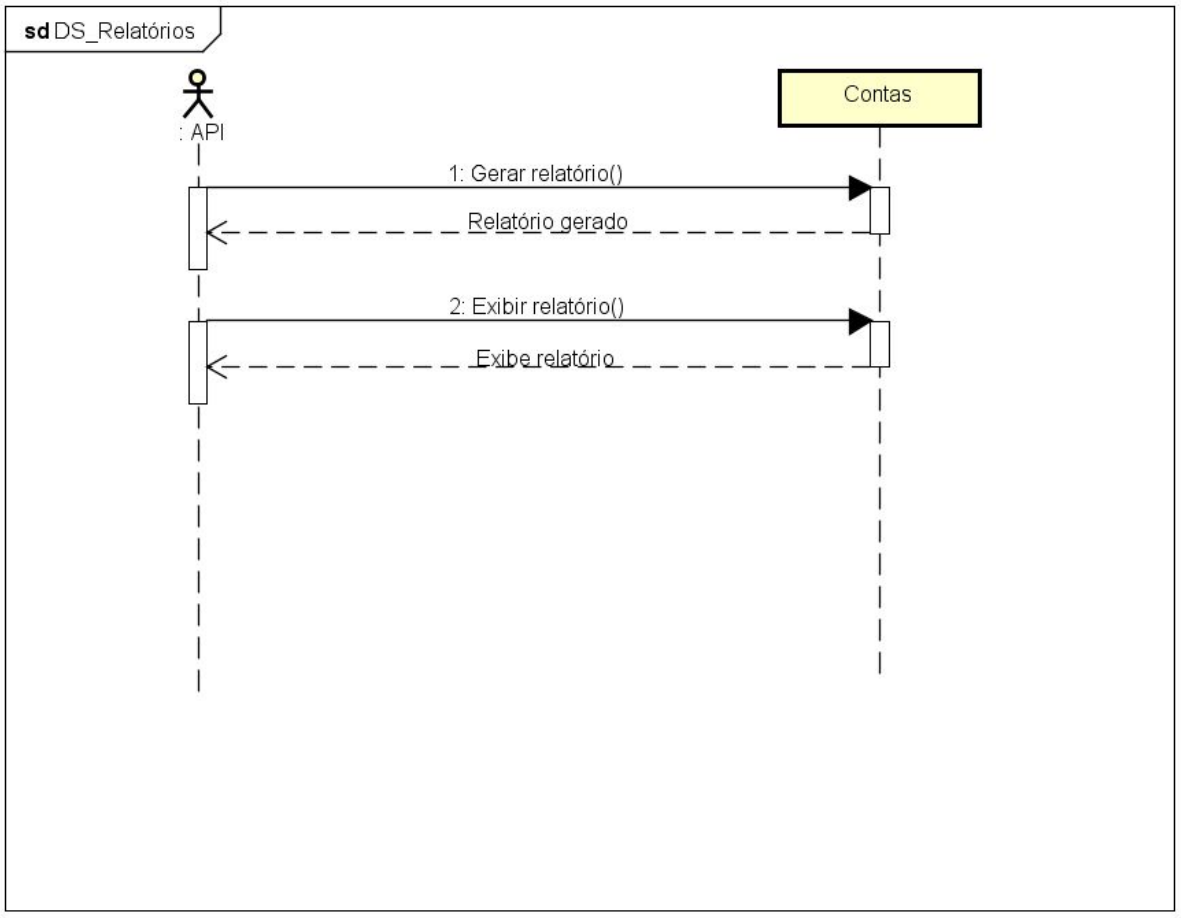
4.1 Realizações de Casos de Uso











5. Visão Lógica

As partes significativas do ponto de vista da arquitetura do modelo de design, são :

Visão de segurança: é responsável pela garantia de que o sistema é seguro, para que o usuário não tenha dúvidas sobre a segurança do sistema ou deixe de usar o sistema por sentir que não é seguro.

Dentro dessa visão há um Subsistema de design de segurança, dentro desse subsistema os pacotes de segurança e dentro deste pacote as classe:

- API : é a classe responsável pelo desempenho do sistema, sendo assim é responsável pela funcionalidade, segurança e eficiência do sistema.

Visão de usabilidade: é responsável pela interface do sistema se mostrar usual ao cliente, e que ele não tenha dificuldade na hora que for usar o sistema.

Dentro dessa visão há um Subsistema de design de usabilidade, dentro desse subsistema os pacotes de usabilidade e dentro deste pacote as classes:

- Interface : é a classe responsável pelo que o usuário vê do sistema, por isso ela é responsável pela usabilidade do sistema.

Visão de confiança: é responsável pela confiabilidade do sistema, assim sendo específica as características que o sistema deve possuir para ser considerado confiável.

Dentro dessa visão há um Subsistema de design de usabilidade, dentro desse subsistema os

pacotes de usabilidade e dentro deste pacote as classes:

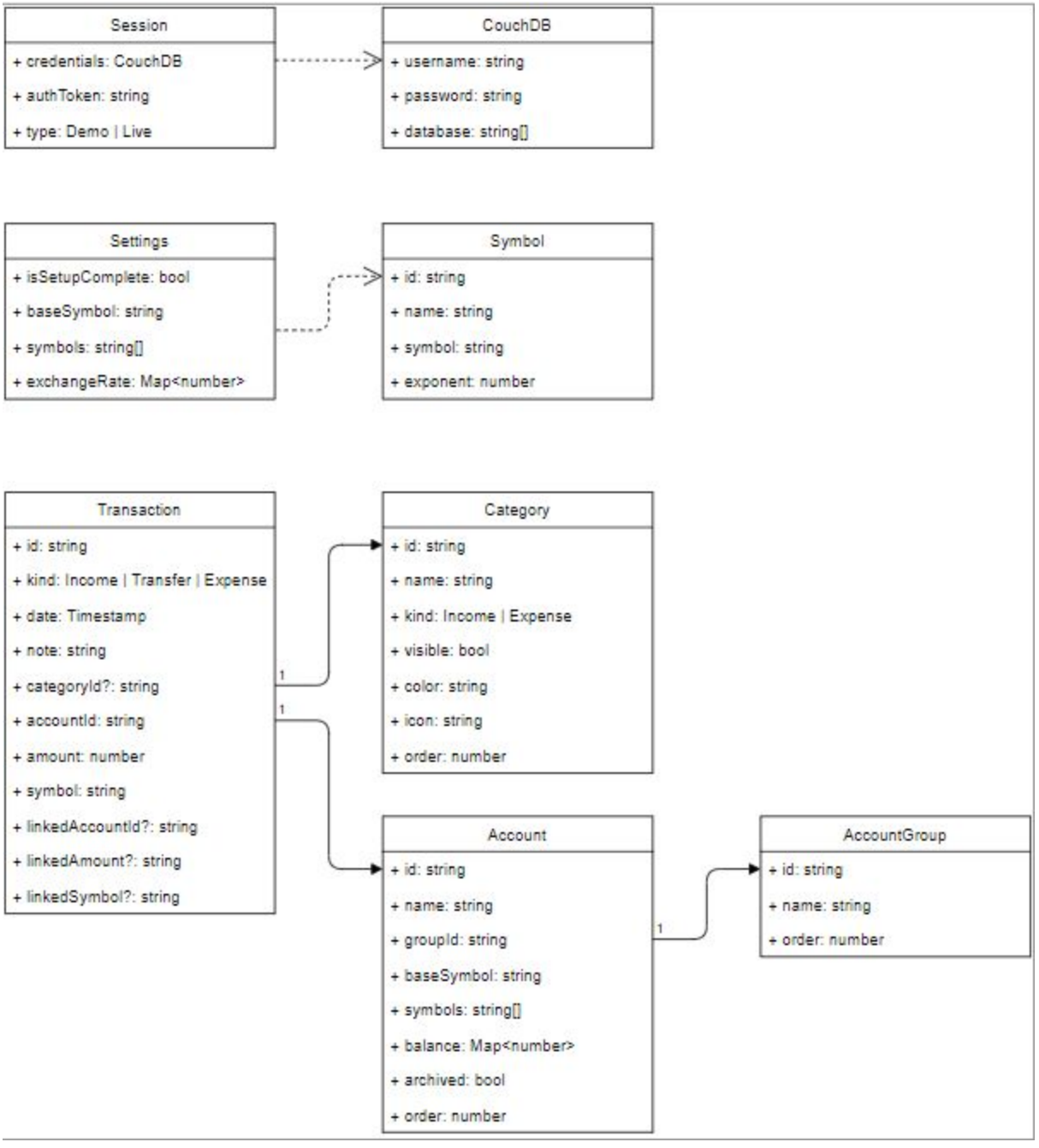
- API : é a classe responsável pelo desempenho do sistema, sendo assim é responsável pela funcionalidade, segurança e eficiência do sistema.

5.1 Visão Geral

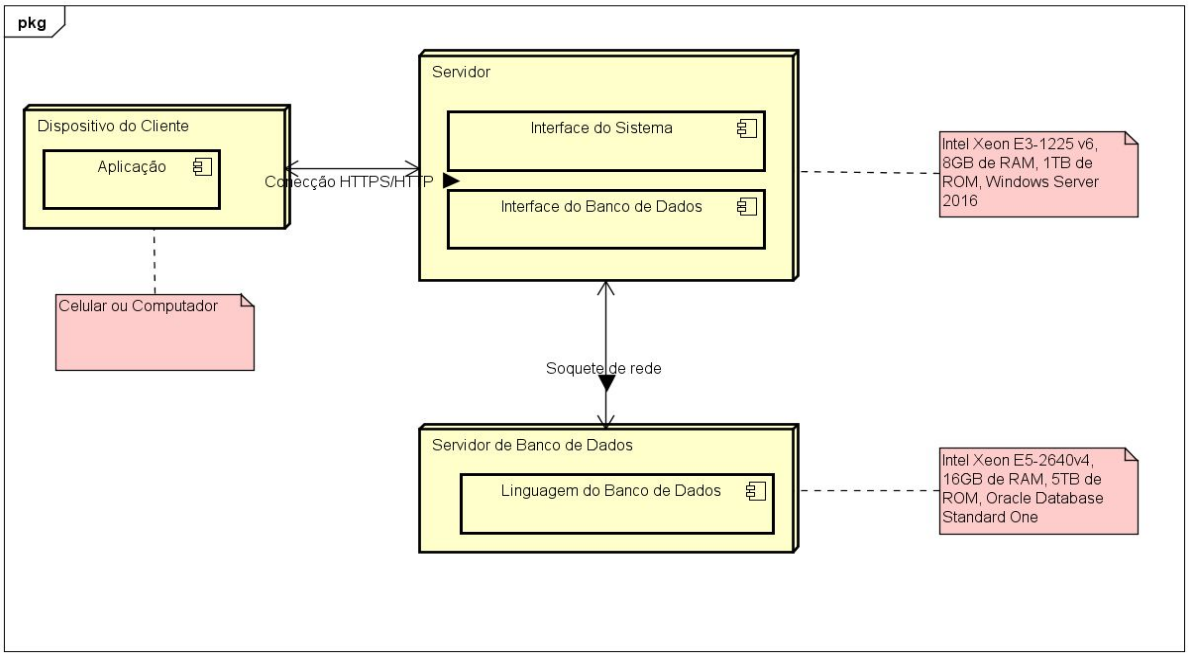
Visão geral do design é baseado na usabilidade, segurança e confiança, desta forma são divididos em três pacotes de nível superior, o pacote da visão de usabilidade, o pacote da visão de confiança e o pacote da visão de segurança.

Cada um deles tem o seu subsistema de design o subsistema da usabilidade, o subsistema da confiança e o subsistema da segurança, dentro de cada subsistema tem estão os pacotes, os quais são, respectivamente : pacote de segurança, que tem a classe API; pacote de usabilidade, que tem a classe Interface; pacote de confiança, que tem a classe Interface

5.2 Pacotes de Design Significativos do Ponto de Vista da Arquitetura



6. Visão de Implantação

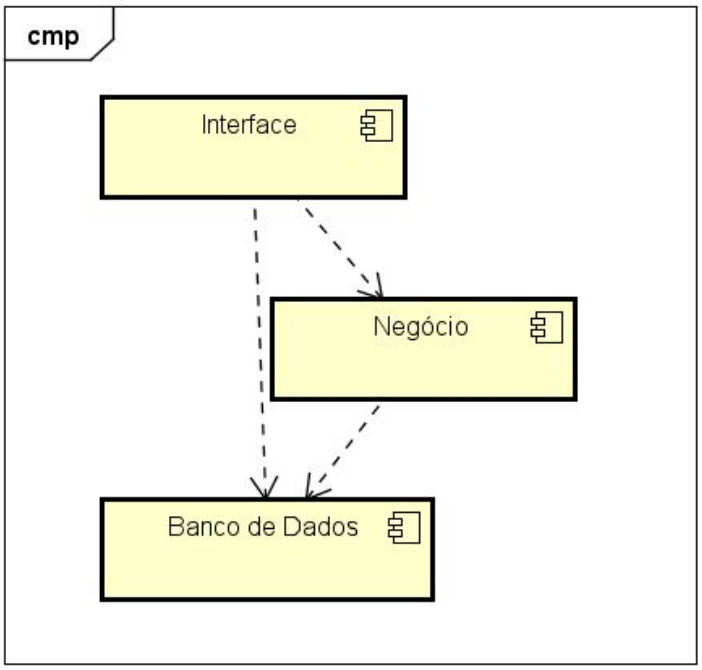


7. Visão da Implementação

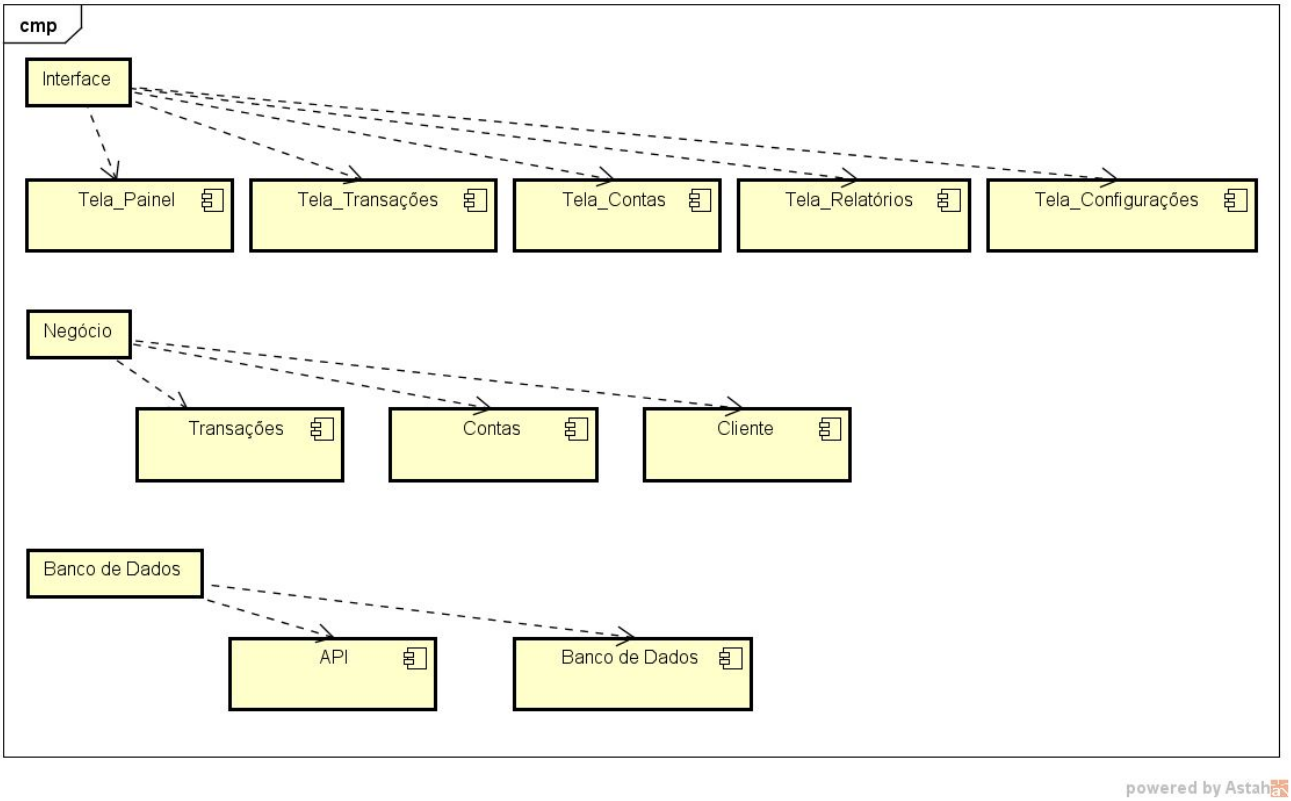
O modelo geral foi divido em 3 camadas, sendo elas: camada de interface, camada de negócio e camada de banco de dados. Os componentes mais significativos são aqueles que possuem total foco do sistema, são aqueles que desempenham a função principal do sistema.

7.1 Visão Geral

- Camada de Interface - Camada de Negócio: onde acontece a interação “principal”, a camada de interface onde o cliente tem a interação com a camada de negócio que é onde fica a aplicação do sistema.
- Camada de Interface - Camada de Banco de Dados: é a interação onde é coletado dados que serão exibidos sem nenhum tratamento, no sistema seria o calendário e o horário.
- Camada de Negócio - Camada de Banco de Dados: é a interação onde acontece a troca de dados, dados salvos são “recuperados” para utilizar durante a interação com o usuário e dados onde dados gerados são armazenados.



7.2 Camadas



8. Tamanho e Desempenho

1) Elementos chave
    a) Transações: 1000 transações registradas por cadastro(pessoa)
    b) Contas: 100 contas por cadastro(pessoa)
2) Medidas de desempenho
    a) Tempo médio de resposta: inferior à 0.1s

b) Taxa de transferência média: 1GB/ Mês
c) Taxa de transferência mínima: 0.1GB/ mês
d) Taxa de transferência máxima: 2GB/ mês

3) Arquivos Executáveis

É uma aplicação Web, então a maior parte do sistema fica na nuvem, o arquivo executável(APP) ficará com o tamanho entre 30MB e 35MB.

9. Qualidade

A arquitetura do software contribui para a portabilidade já que foi definido em quais plataformas o sistema irá funcionar, no caso nas principais plataformas atuais, suas necessidades de hardware também são descritas no documento. Em relação à extensibilidade, o software pode ser atualizado enquanto está em execução já que é um sistema em tempo real. A própria qualidade recebe contribuição da arquitetura, o sistema não terá nenhum tempo de inatividade não programada, ou seja, o sistema não terá quedas não programadas.