ABEX III - PROJETO DE SOFTWARE- 1030784

2023-01

| **Professor(a)(es)** | RADAMÉS PEREIRA  VIVIANE DUARTE BONFIM |
| --- | --- |

Atributos de uma boa especificação de Projeto de Software:

* Clareza
* Não Ambígua
* Completa
* Simples
* Bem escrita

**UNIVERSIDADE COMUNITÁRIA REGIONAL DE CHAPECÓ - UNOCHAPECÓ**

**ÁREA DE CIÊNCIAS EXATAS E AMBIENTAIS**

**SISTEMAS DE INFORMAÇÃO/CIÊNCIA DA COMPUTAÇÃO**

Aplicação para controle financeiro

Equipe: **Lucas Ribacki**

**Introdução**

* 1. ***Resumo do Projeto***

Aplicação web, com a idéia de controle de financeiro, onde cada usuário terá uma ou mais contas pessoais, podendo acessar de qualquer lugar com internet, tem meios de inserir informações e ter disponíveis várias funcionalidades na aplicação que proporcionará por meio dessas funcionalidades fazer simulações a futuro, análise do estado atual, entre outros, categorização de gastos informado pelo usuário ao decorrer do uso, podendo gerar relatórios não só de gastos mas ganhos a futuro.

* 1. ***Plataforma de desenvolvimento***

Aplicação web, sendo backend e frontend em Ruby on Rails e banco de dados PostgreSQL

* 1. ***Plataforma de operação***

A aplicação será locada em um servidor público, onde estará ao alcance do usuário fazer seu acesso, como é web também será viável o uso mobile, por mais que não tenha um foco definido no quesito mobile, como a aplicação está na nuvem ela fica ao seu alcance.

* 1. ***Definições e siglas***

Não se aplica.

* 1. ***Perspectiva do produto***

Propósito de possibilitar o melhor controle financeiro tanto pessoa física como jurídica, pois com as funcionalidades de perspectiva dispostas pode-se avaliar gastos e rendas e gerar relatórios independente do tipo do cliente.

Utilizando a aplicação web, temos à disposição do cliente a possibilidade de acesso de qualquer lugar que possua internet, seja desktop ou mobile.

Com a possibilidade de categorizar seus gastos ou rendas em categorias já existentes ou criadas a partir de uma necessidade e controle específico, permite ao usuário uma maior mobilidade em seu controle, e não engessado a algo pré determinado.

Focando em uma ideia de gerar relatórios para comprovações ou comparações, tem-se à disposição ferramentas moldáveis, filtros de busca ou filtros de comparações, fazendo que capare períodos e com isso perspectivas e situações.

* + 1. **Modos de operação**

Desenvolvimento web, com um back end e frontend.

* + 1. **Requisitos de adaptação ao ambiente**

Não se aplica.

* 1. ***Funções do produto***

**Controle mensal**

Expõem o gasto e lucro do mês, conectando e separando por categoria ou grupo personalizado, fazendo um cálculo simples e mostrando ao cliente a sua situação atual do mês.

**Separação de contas por categoria**

Através de agrupamentos, mostrando ao cliente com mais clareza onde está sendo movimentado suas finanças.

**Filtro predefinido ou personalizado**

O cliente deve se atentar a algo específico para ele, fazendo assim que o sistema calcule a partir de seu filtro, tando suas receitas ou despesas.

**Filtros de comparação de períodos**

Usuários informa duas datas que são o primeiro período, e outras duas datas que são o segundo período, assim a aplicação gera um relatório em paração de gastos e rendas.

**Gerar PDF ou EXCEL**

A Partir de todos os filtros tem-se a possibilidade de exportar informações para PDF ou Excel.

**Alteração de dados já existentes**

Toda informação inserida na aplicação tem a possibilidade de ser excluída ou alterada, mas gerando um histórico para certificações futuras.

* 1. ***Características dos usuários***

O usuário terá uma ou mais contas, com diferentes logins, podendo ser um empreendedor que necessita de uma facilidade em gerenciar seus investimentos e renda para montar estratégias para o futuro ou um usuário que necessite economizar e gerir seus gastos, observando no que está gastando.

Também um usuário com uma idade avançada que quer ter maior controle sobre sua renda fixa mensal, estipulando metas para compras e facilitar investimentos ou uma melhor forma de como poupar seu dinheiro.

* 1. ***Restrições***

Lei de proteção de dados.

* 1. ***Hipóteses de trabalho***

Trabalhará com o versionamento de branchs no github, uma branch que irá estar rodando em produções, outras para seu desenvolvimento. Tem-se a liberdade de fazer diversas modificações e testá-las das mais diversas maneiras, para ter a garantia de qualidade.

Após ter a confiabilidade pode-se fazer o merge da branch de desenvolvimento para branch de produção.

1. **Requisitos específicos**
   1. ***Interfaces externas***
      1. **Visão geral**

O usuário cria sua respectiva conta com login e senha, seguidos de mais informações, após essa etapa ele insere dados que serão usados mais tarde, como quanto ganha no mês e seus gastos fixo.

Após essa inserção de dados, o usuário pode solicitar algumas funcionalidades do sistema que ao ser solicitado, retorna as respectivas requisições.

* + 1. **Requisitos para interfaces gráficas de usuário**

Interface de login:



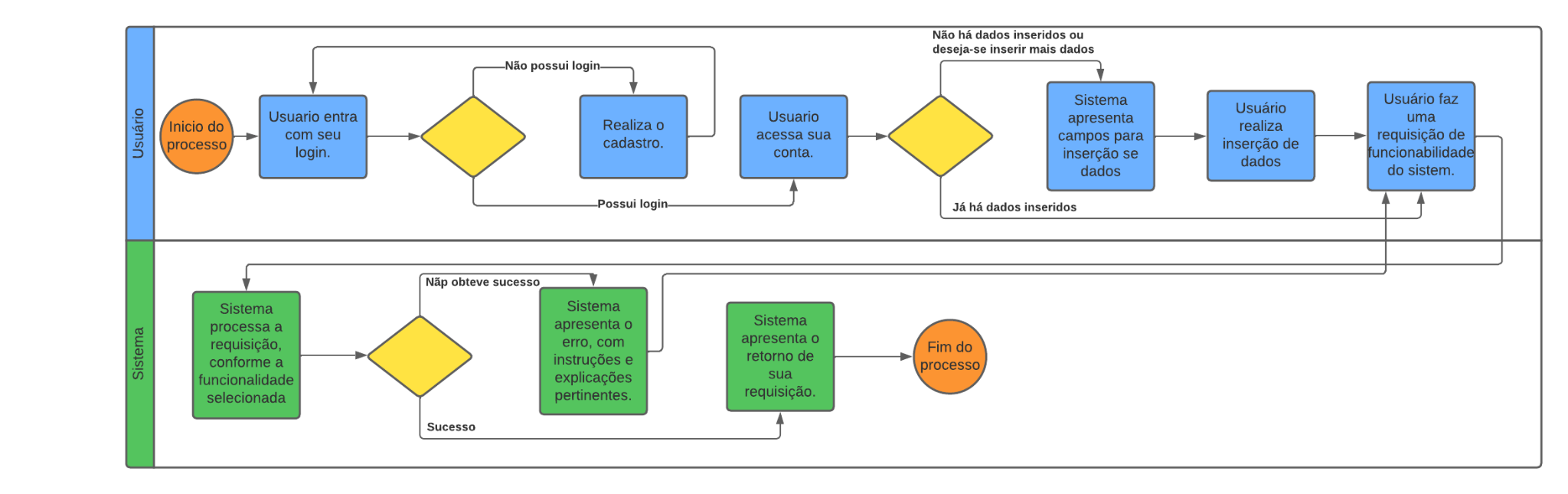
Interface de tela inicial:

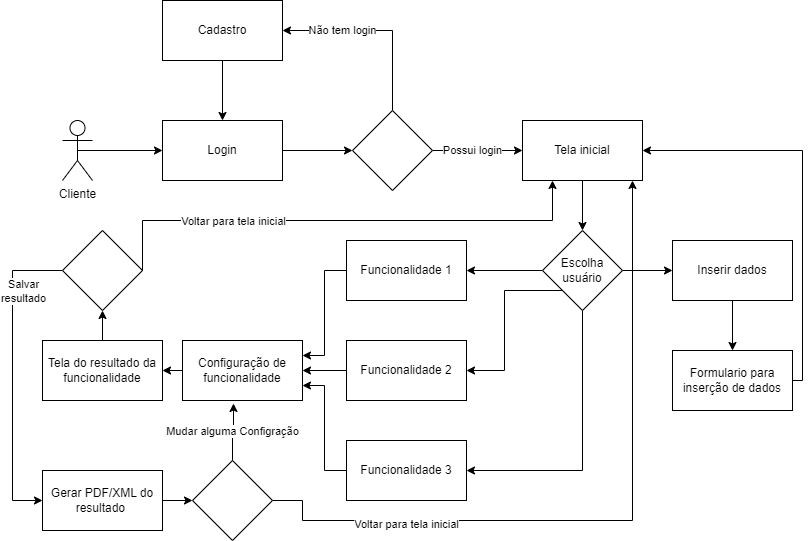


* Uma descrição dos relacionamentos com outras interfaces;

As interfaces possuem seu início com o login, após entramos para com a tela inicial, nela temos acesso a todas as ferramentas da aplicação. Após cada inserção de dados voltaremos para a tela inicial, para dar sequência ao que fazermos com os dados inseridos.

* Um diagrama de estados/atividades, caso necessário para melhor entender-se o comportamento requerido da interface;

****



* Uma lista dos campos de dados da interface;

Nas interfaces temos:

Inputs para inserção de dados;

Labels para informações dos respectivos campos;

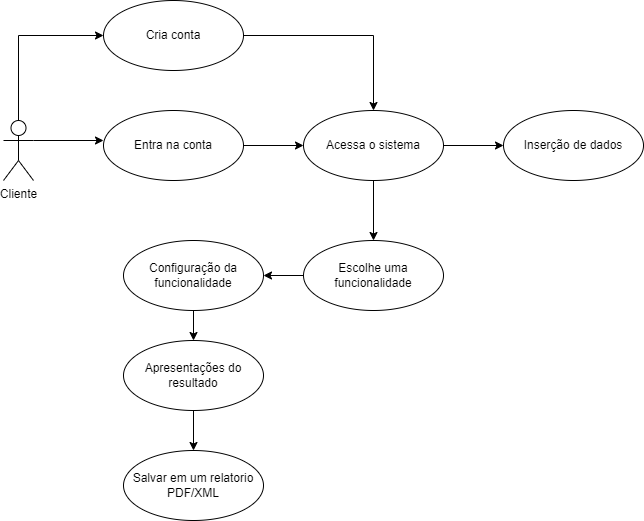
Títulos para informações de agrupamentos de dados;

Texto com links para acessar mais informações sobre o item;

* Uma lista dos comandos da interface;

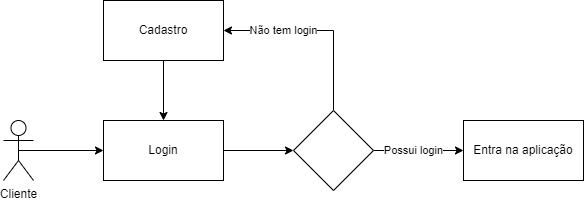
Não se aplica.

* BPM;
  1. ***Requisitos funcionais***
     1. **Diagramas de casos de uso**



Incluir todos os casos de uso que se pretende implementar em uma liberação. Pode-se incluir ainda: um certo caso de uso e seus relacionamentos, todos os casos de uso para um certo ator.

* + 1. **Fluxos dos casos de uso**
* Pré-condições para a realização do caso de uso;



* Fluxo principal do caso de uso (sucesso), descrito na forma de uma sequência de passos;

Está demonstrada acima.

* Fluxos alternativos do caso de uso;
* Descrições mais formais, como diagramas de estado ou de atividade, se a complexidade do caso de uso exigir;
* Observações.
  1. ***Requisitos não-funcionais***
     1. **Requisitos de desempenho**

Desempenhos ofertados pelo servidor onde a aplicação for locada, mas um exemplo muito plausível é o site Hostinger.com, seu plano mais popular oferece:

200 GB de Armazenamento SSD;

Backups diários;

1,50 GB RAM;

2 Núcleo de CPU;

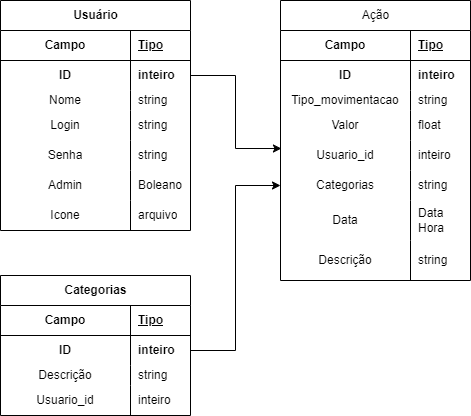
Largura de Banda Ilimitada

Domínio Grátis;

SSL Ilimitado Grátis.

* + 1. **Requisitos de dados persistentes**

Conta-se com dados do usuário e dados da ação que o mesmo realiza, conectando os dois pelo id do usuário, separando todas as ações pelo usuário que as cria, data que é realizada a ação e sua categoria que possui

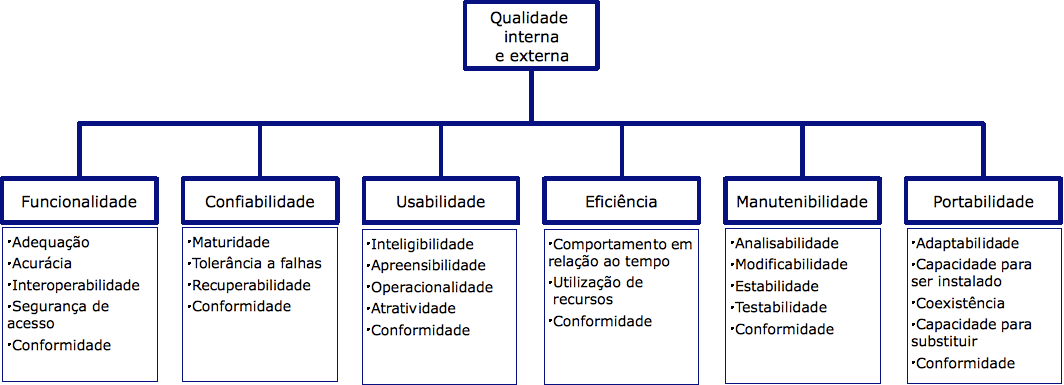


* + 1. **Restrições ao desenho**

Restrições de projeto impostas por padrões externos, com influência da legislação..

* + 1. **Atributos de Qualidade**

Seguindo a [ISO-9126](https://drive.google.com/open?id=14-OfxrvT5pD4sblFt234r8NPBNwt2HmG), com a falta de usabilidade de alguns atributos, temos com exemplo:



No processo de desenvolvimento, terá a separação de partes complexas de desenvolvimentos, separando elas em subtarefas, no jira software. O que é importante salientar, é o respeito para com boas práticas requisitadas pela linguagem e seu framework, neste caso a linguagem Ruby e Ruby on Rails framework.

Para nosso produto temos os seguintes atributos internos e externos:

**Funcionalidade**

**Adequação** para o usuário, dispondo de ferramentas ajustáveis, sempre com algo pré definido mas permitindo alterações em seus parâmetros e agrupamento de filtros para casos específicos para os mais diversos usuários.

**Acurácia** de seus relatórios, contando com a configuração dos usuários para ter o mais precisão em suas necessidades.

**Segurança** de dados, pois as informações são de extrema importância e sensibilidade por se tratar da condição financeira.

**Conformidade** de seguir todas as leis impostas na proteção de dados, oferecendo aos usuários um meio de se sentir seguro tendo suas informações protegidas.

**Confiabilidade**

**Maturidade** de software, promovendo os mais exaustivos testes em relação a recursos existentes e novos, evitando falhas e enganos para com o produto.

No caso de alguma falha inesperada, temos **tolerância a falhas**, onde para cada parte do sistema tem-se maneiras de contorná-las, usando métodos que o próprio framework disponibiliza.

**Recuperabilidade** das informações no momento do erro é tratado pelo start transaction disponibilizado pelo framework, sendo ele interrompe o fluxo de dados do branco de dados, impossibilitando seu acesso em caso de erro.

**Conformidade** para com leis e exigências postas ao tipo de informação que está se trabalhando.

**Usabilidade**

**Inteligibilidade** e **Apreensibilidade** para com o usuário, interface simples e auto explicativos, campos de funcionalidades com “caixinhas” de informações, sendo ela presentes em várias partes do software.

Para cada ação do usuários, tem-se um histórico do que foi criado/alterado/excluído, implicando diretamente a **proteção frente a erros de usuários**, que ao errar o usuário tem um histórico para corrigir ou mesmo conferir se foi certo a ação.

**Estética/Atratividade** do sistema conta com uma interface gráfica simples de forma a proporcionar para o usuário uma forma mais limpa de se orientar no sistema tendo as cores configuráveis pelo próprio usuário.

**Eficiência**

**Comportamento em Relação ao Tempo** do sistema é boa nos mais diversos cenários pois sua busca e comparação é quase que total por meio do banco de dados, fazendo buscas e relações no próprio banco, causando uma rapidez nas requisições.

**Manutenibilidade**

**Analisabilidade** de erros no código é facilmente visível, pela separação de todas as operações implementadas no software, sendo o local do problema filtrado imediatamente por esta separação.

**Estabilidade** para com a arquitetura do planejamento de todas as partes do software, separando com digo, mas cada conexão comentada ou documentada, assim evitando efeitos colaterais em lógicas que podem influenciar em diversas partes do sistema.

**Testabilidade** de todas as partes feitas com o TDD disponível de forma nativa pelo framework, de foram que além de testes com dados fictícios, possa testar partes do código com lógicas do próprio código, com retorno de erros e ainda bugs que não seriam pegos em teste utilizando o sistema rodando como forma de teste.

**Portabilidade**

Sistema facilmente acessado pelo usuário, pois se trata de uma aplicação web. Somente com o empecilho de não ser acessível quando não tem conexão com a internet.

**Qualidade em Uso**

**Eficácia** do software em disponibilizar as requisições que serão feitas pelo usuário e as apresentar de forma que atinja a satisfação do usuário em seus resultados.

**Produtividade** proposta para com os usuários é sempre dispor de uma forma para com o mesmo tenha uma certeza e tenha mais confiança e planejamento para seu uso com o software.

**Segurança** com os dados que são inseridos é passado para o usuário por meio de explicações e arquiteturas para com o seu desenvolvimento primário, já levando em consideração.

**Satisfação** do cliente em ter seu controle financeiro, de maneira que lhe traga uma segurança em seu dia a dia, para ter sempre a mão o software.

1. **Análise de UCP**

As tabelas de escopo de valor do produto e tempo de desenvolvimento com Use Case Points - UCP.

1 – Contar os atores e identificar sua complexidade;

2 – Contar os casos de uso e identificar sua complexidade;

3 – Calcular os PCUs não ajustados;

4 – Determinar o fator de complexidade técnica;

5 – Determinar o fator de complexidade ambiental;

6 – Calcular os PCUs ajustados;

Referências:

*IEEE Std. 830 – 1993. IEEE Recommended Practice for Software Requirements Specifications.*

*IEEE ISO/IEC/IEEE 29148 – 2011. IEEE Systems and software engineering — Life cycle processes — Requirements engineering*

**OBSERVAÇÃO: Os itens deste modelo de especificação, recomendado pela IEEE, poderão ser complementados com novos itens caso sejam justificáveis.**