Documentação de Software

Food Manager

Autores:

Vitor Cristiano Fellizatti

Thiago dos Santos Rodrigues

Thiago Barros Gomes

Vinicius Augusto Nigra

Vitor de Carvalho

Documento de especificação de requisitos do projeto

" Food Manager."

Versão: 1.2

Projeto Interdisciplinar do 3º semestre do curso de Desenvolvimento de Software Em Multiplataforma desenvolvido na Faculdade de Tecnologia de Araras (FATEC), apresentado como Trabalho de Conclusão.

Faculdade de Tecnologia de Araras

DESENVOLVIMENTO DE SOFTWARE EM MULTIPLATAFORMA

TRABALHO DE CONCLUSÃO DO 3ºSEMESTRE.

Orientadores:

Prof. **RENATO CIVIDINI MATTHIESEN**

Gestão Ágil de Projeto de Software

_____

Prof. Orlando Saraiva

Desenvolvimento Web III

Prof. **THIAGO GONCALVES MENDES**

Banco de Dados Não Relacional

SUMÁRIO

1.0 INTRODUÇÃO

Este documento tem por propósito especificar os requisitos necessários da página de web do Food Manager para o seu desenvolvimento. Além de servir de referência para a manutenção do software. Foram utilizados diagramas UML (diagrama de casos de uso e diagrama de sequência), requisitos funcionais e requisitos não funcionais para ajudar no entendimento das funcionalidades do sistema.

1.1OBJETIVO

Nosso sistema permite a organizações de caridades e banco alimentos.

Registrando e rastreando doações, estoques e distribuições de alimentos.

Várias instituições recebem produtos com variedades de validades e como o gerenciamento é humano acaba ocorrendo falhas pois sendo manual via papel de controle os alimentos acaba sendo mal distribuídos. Através do nosso sistema vai ser mais fácil o controle dos produtos os que tem prioridade de saída e os tem que maior necessita podem ser pegos e distribuídos primeiros

1.2 **ESCOPO**

Gerenciador de Armazenamento de alimentos para pessoas com necessidades - F.M

Executável: Nesse projeto faremos a ponte entre usuário que precisam de alimentos com os doadores.

Benefícios: Agilizar informações necessárias, compartilhar de forma rápida o estado.

Objetivos: Agilizar a conexão do usuário que precisa com o produto e quem disponibiliza o mesmo,

Meta: Acelerar o encontro entre essas duas entidades.

1.3 VISÃO GERAL

- Muitas instituições lidam com uma variedade de produtos com datas de validade diversas. O gerenciamento manual desses alimentos resulta frequentemente em falhas, levando a distribuições inadequadas. Através do nosso sistema, buscamos resolver esse problema, proporcionando:
- **Controle de Inventário Eficiente**
- **Priorização de Alimentos**
- **Agilidade no gerenciamento da distribuição**

1.4 DESCRIÇÃO GERAL

Os usuários conseguiram colocar as informações que serão necessárias para conseguimos fazer a ponte em quem precisa com aqueles que irão doar.

Terão janelas que vai ter essa informações de são necessária para a facilida.

2.0 REQUISITOS FUNCIONAIS

RF01: Cadastrar de informações - obrigatório

RF02: Compartilhar de informações entre usuários da plataforma - obrigatório;

RF03: Acrescentar informações - obrigatório;

2.1 REQUISITOS NÃO FUNCIONAIS

RNF01: tempo de resposta não deve exceder 5 segundos;

RNF02: Acesso de internet;

RNF03: Python como linguagens;

RNF04: Mongo DB como banco de dados;

RNF05: Utilização de framework.

2.2 REQUISITOS DE INTERFACE

RI01: Layout Intuitivo e fácil de navegar. Com elementos claros e organizados, permitindo que os usuários encontrem rapidamente as informações necessárias.

RI02: Design Responsivo: Acessível em diferentes dispositivos, como desktops, tablets e smartphones.

RI03: Segurança e Privacidade: A segurança dos dados do usuário. A interface deve implementar medidas robustas de segurança, como criptografia e autenticação.

RI04: Registro e Pesquisa Eficiente: A interface deve permitir o registro eficiente de informações.

2.3 ATRIBUTOS DE QUALIDADE

AQ01: Segurança e Privacidade: A segurança dos dados é fundamental. Implementação de medidas robustas de segurança, como criptografia e autenticação, para proteger as informações confidenciais dos pacientes contra acessos não autorizados.

OBRIGATÓRIO

AQ02: Design Responsivo: O Food Manager é acessível em diferentes dispositivos, como desktops, tablets e smartphones. O design responsivo garante que a interface se ajuste automaticamente ao tamanho da tela, proporcionando uma experiência consistente e adaptável em todos os dispositivos.

DESEJÁVEL

2.4 CARACTERISTICAS DOS USUÁRIOS

Este *software* destina-se a facilitar o encontro com a instituição que tem o produto para quem necessita do mesmo.

INSTITUIÇÃO: Será a entidade que seria empresas e comércio quem excedente de produto que queira doa esse excedente.

PESSOAS: Essa entendida será os usuários que receberam as doações.

3.0 ANEXO

Métodos utilizados:

Brainstorming: Utilizamos para levantar ideias e dar início a uma discussão que ainda não está formada. Assim rapidamente se levanta uma grande quantidade de ideias.

Para sabermos a necessidade de quem vai utilizar o produto ou saber detalhes sobre o funcionamento de um processo qualquer. Coletando assim as informações como são, permitindo aferir a usabilidade de um processo.

Pensando em grupo, com participantes que podem ser internos ou externos ao projeto. No *brainstorming*, os participantes falam livremente, sem coibição.

Protótipos: Utilizamos para coletar um *feedback* mais concreto das partes interessadas. Sendo atualizado diversas vezes, até solidificar os requisitos.

Um subproduto da solução principal que deve mostrar as funcionalidades que se desejem avaliar durante o levantamento de requisitos de um projeto.

4.0 FERRAMENTAS

FER01: *Visual Studio Code* – codificação

FER02: *MongoDB* – banco de dados

FER03: *Zeal* – documentação

FER04: MongoDB - documentação

FER05: *Draw.io*

4.1 LINGUAGENS

LING01: *JavaScript*

LING02: HTML

LING03: CSS

LING04: PHP

LING5: NOSQL

LING6: Python

5.0 DIAGRAMA DE CASO DE USO

5.1 INSTITUIÇÃO

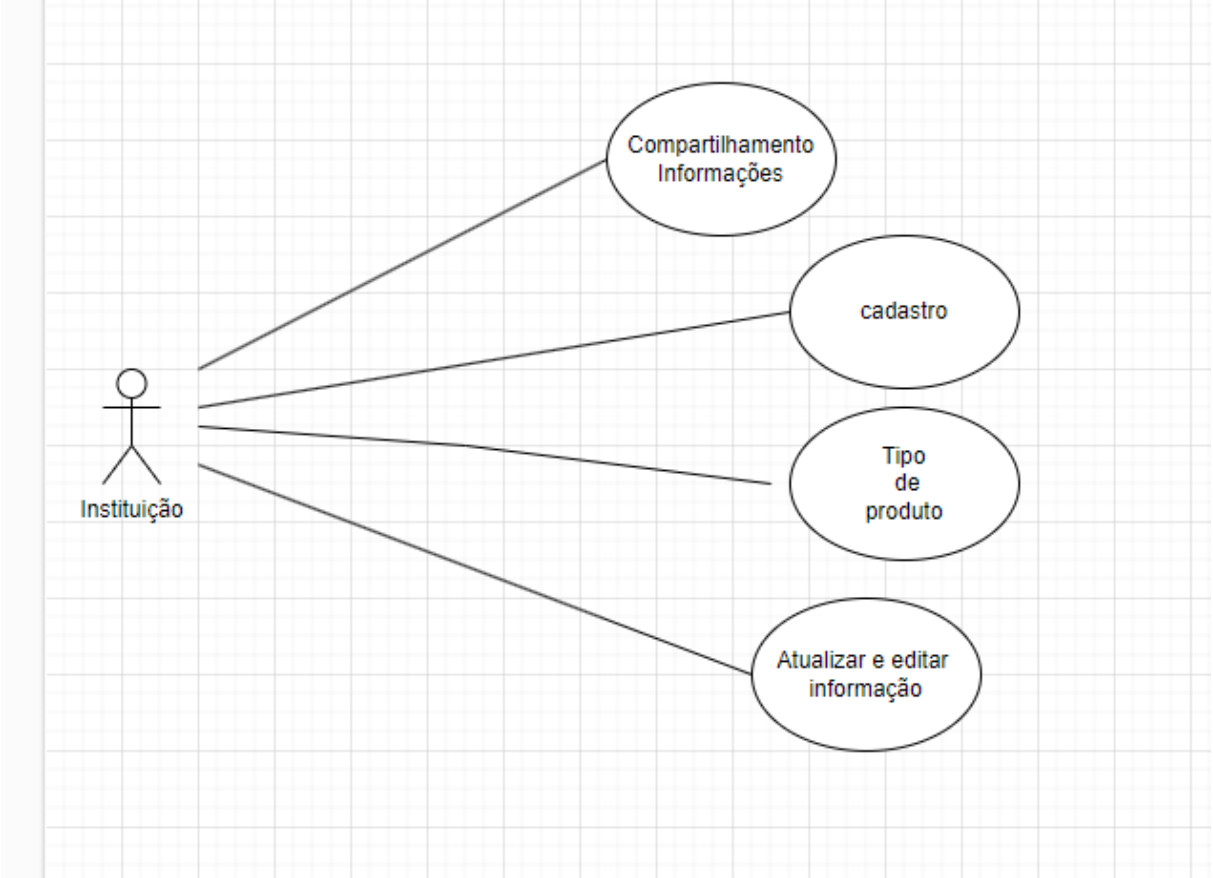

Figura 01 – Diagrama de Caso de Uso

6.0 BANCO DE DADOS

6.0.1 Banco de dados NoSQL:

Um banco de dados não relacional, também conhecido como NoSQL (Not Only SQL), é um tipo de sistema de gerenciamento de banco de dados (SGBD) que difere dos bancos de dados relacionais tradicionais em sua abordagem de armazenamento e recuperação de dados. Aqui estão algumas características e uma breve explicação de bancos de dados não relacionais:

6.0.2 Modelo de Dados Flexível:

Diferentemente dos bancos de dados relacionais que seguem um modelo de dados tabular, os bancos de dados NoSQL podem adotar diversos modelos de dados, como documentos, chave-valor, grafos ou famílias de colunas. Isso oferece flexibilidade para lidar com diferentes tipos de dados e requisitos.

6.0.3 Escalabilidade Horizontal:

Muitos bancos de dados NoSQL são projetados para escalar horizontalmente, o que significa que você pode adicionar mais servidores para lidar com um aumento na carga de trabalho. Isso contrasta com a abordagem vertical dos bancos de dados relacionais, onde você geralmente precisa aumentar a capacidade de um único servidor.

6.0.4 Sem Esquema Fixo:

Os bancos de dados NoSQL geralmente não têm um esquema fixo, permitindo que você adicione novos campos aos documentos (ou equivalentes) sem a necessidade de modificar esquemas existentes. Isso é útil em ambientes onde os requisitos de dados estão em constante evolução.

6.0.5 Desempenho em Leitura e Gravação:

Muitos bancos de dados NoSQL são otimizados para operações específicas, como leitura ou gravação, tornando-os eficientes em cenários onde essas operações são predominantes. Isso é especialmente útil em aplicativos com grandes volumes de dados e alta concorrência.

6.0.6 Consistência Eventual:

Alguns bancos de dados NoSQL adotam o modelo de consistência eventual, o que significa que, em determinadas circunstâncias, pode haver uma discrepância temporária entre os dados em diferentes nós do sistema. Isso é aceitável em cenários onde a consistência imediata não é uma prioridade absoluta.

6.0.7 Aplicações Distribuídas e Big Data:

Bancos de dados NoSQL são frequentemente usados em ambientes distribuídos e em cenários de big data, nos quais o volume de dados é grande e a distribuição geográfica dos usuários ou servidores é uma consideração importante.
Exemplos de bancos de dados NoSQL populares incluem MongoDB (documentos), Cassandra (colunas), Redis (chave-valor), e Neo4j (grafos). A escolha do banco de

dados NoSQL dependerá dos requisitos específicos do projeto e das características desejadas para manipulação de dados.

6.1 EXPLICAÇÃO DE FUNCIONAMENTO DO SISTEMA

6.1.1 Collection (Coleção):

Uma coleção no MongoDB é análoga a uma tabela em bancos de dados relacionais. No entanto, ela é mais flexível, pois não exige que todos os documentos (linhas) tenham a mesma estrutura. Cada coleção contém um grupo de documentos MongoDB relacionados. Em seu projeto de alimentos, por exemplo, você pode ter uma coleção chamada "produtos" que armazena informações sobre diferentes produtos alimentícios.

6.1.2 Documentos:

Documentos são os objetos individuais armazenados em uma coleção. Em termos simples, um documento é como uma linha em uma tabela de banco de dados relacional. No MongoDB, os documentos são representados em formato BSON (Binary JSON), que é uma extensão do formato JSON. Cada documento contém pares de chave-valor, onde as chaves são os nomes dos campos e os valores são os dados associados. Por exemplo, um documento em sua coleção "produtos" pode representar um item específico, contendo informações como nome, categoria, data de validade, etc.

6.1.3 Atributos:

Os atributos são os campos ou propriedades individuais dentro de um documento. Em um contexto mais amplo, eles são as chaves em pares de chave-valor dentro do documento. Cada atributo contém um valor associado, que pode ser um número, uma string, um objeto ou até mesmo outra coleção. No seu projeto, os atributos de um documento na coleção "produtos" podem incluir coisas como "nome", "categoria", "data de validade", etc.

6.2 EXPLICAÇÃO DE FUNCIONAMENTO DO SISTEMA

Meu projeto em Python utiliza MongoDB com a biblioteca PyMongo para criar e gerenciar um banco de dados de alimentos. Imagine este banco de dados como um grande armazém virtual onde armazenamos informações sobre diferentes produtos alimentícios.

Quando um usuário deseja adicionar um novo produto ao sistema, ele fornece detalhes como nome, categoria, data de validade, etc. Essas informações são então registradas em uma área específica do banco de dados chamada "produtos".

A parte interessante ocorre quando outra pessoa, um requerente, expressa o desejo de adquirir um produto específico. Nesse momento, o sistema entra no banco de dados, localiza o produto desejado na coleção de produtos e o transfere para a coleção do requerente.

Essa ação de transferência cria uma associação entre o produto e o requerente, indicando que aquele produto agora está vinculado a essa pessoa em particular. Dessa forma, o sistema consegue rastrear quem possui quais produtos e facilita a gestão do inventário de alimentos.

Em resumo, o projeto utiliza MongoDB para criar uma base de dados de alimentos, permite a adição de novos produtos e, quando solicitado, move esses produtos da coleção geral para a coleção do requerente, estabelecendo uma conexão entre a pessoa e os produtos que ela deseja adquirir.

6.2.1 EXEMPLO DE COLLECTIONS

6.2.1.1 Nossa collection Produtos

```
_id: ObjectId('656f085fc0dd6e7dd306906c')
nome: "Feijão"
quantidade: 5
validade: 2023-12-31T00:00:00.000+00:00
```

```
_id: ObjectId('656f2fcf9369d7bbbba3dfa0')
nome: "Macarrão"
quantidade: 2
validade: 2023-12-28T00:00:00.000+00:00
```

```
_id: ObjectId('656f7bb7e5d61eb364071913')
nome: "Leite em pó"
quantidade: 10
validade: 2024-01-01T00:00:00.000+00:00
```

6.2.1.2 Nossa collection Requerentes

```
_id: ObjectId('656f9609703eb5a9fead4a7d')
nome: "Thiago"
email: "thiago@hotmail.com"
telefone: "19998256345"
alimento: "Feijao"
```

```
_id: ObjectId('656f961c0afbd16f65795113')
nome: "Thiago"
email: "thiago@hotmail.com"
telefone: "19998256345"
alimento: "Feijao"
```

6.2.1.3 Nossa collection Requerentes com agregação efetuada

```
_id: ObjectId('656f179c02210e1a1af60331')
nome: "Thiago Barros"
email: "thiago@gmail.com"
telefone: "19998563214"
alimento: "Arroz"
produto: Object
  _id: ObjectId('656f7d24b2ccde97f59c34a7')
  nome: "Arroz"
  quantidade: 10
  validade: 2024-01-01T00:00:00.000+00:00
```

6.2.1.4 Aqui um exemplo de comando executado via mongosh para o resultado.

```
FoodManager > db.Requerentes.aggregate([
              {
              $match: {
                _id: ObjectId("ID_EXEMPLO")
              }},
              {
              $lookup: {
                from: "Produtos",
                localField: "alimento",
                foreignField: "nome",
                as: "produto"
              }},
              {
                $unwind: "$produto" }, { $set: { "produto.nome": "Feijão" } }, {
                $merge: { into: "Requerentes", whenMatched: "merge", whenNotMatched: "insert" } }
              ])
```