INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO NORTE DE MINAS GERAIS
***Campus* Montes Claros**

MocFrota
Documento de Arquitetura de Software

Versão 1.5

Histórico da Revisão

Data	Versão	Descrição	Autor
15/05/2023	1.0	Elaboração do Documento de Arquitetura de Software	Emilly Lacerda, Fernanda Pimenta, Thiago Evangelista
16/05/2023	1.1	Continuação da Elaboração	Emilly Lacerda, Fernanda Pimenta, Thiago Evangelista
17/05/2023	1.2	Continuação da Elaboração	Emilly Lacerda, Fernanda Pimenta, Thiago Evangelista
18/05/2023	1.3	Continuação da Elaboração	Emilly Lacerda, Fernanda Pimenta, Thiago Evangelista
26/05/2023	1.4	Continuação da Elaboração	Emilly Lacerda, Fernanda Pimenta, Thiago Evangelista
28/05/2023	1.5	Continuação da Elaboração	Emilly Lacerda, Fernanda Pimenta, Thiago Evangelista

Índice Analítico

Documento de Arquitetura de Software

1. Introdução

Este documento apresenta a arquitetura de software de um sistema de gestão de frota de veículos para uma empresa. Ele descreve os principais aspectos da arquitetura do sistema, incluindo a visão geral, a representação arquitetural, as metas e restrições da arquitetura, a visão de casos de uso, a visão lógica, a visão de processos, a visão de implantação, a visão de implementação, a visão de dados, o tamanho e desempenho e a qualidade.

1.1 Finalidade

A finalidade deste documento é documentar e comunicar a estrutura e a organização do sistema de software MocFrota, que é um software de gestão de frota de veículos. Ele tem o objetivo de fornecer à empresa contratante uma visão geral da arquitetura do sistema, descrever os principais componentes, suas interações, decisões de design e as tecnologias utilizadas.

1.2 Escopo

O documento de arquitetura de software serve como uma referência para os desenvolvedores, analistas e demais stakeholders envolvidos no projeto. Ele fornece orientações, diretrizes e informações cruciais para o desenvolvimento, manutenção e evolução do software. Além disso, auxilia na tomada de decisões técnicas, fornecendo uma base sólida para o planejamento e a alocação de recursos. Ele também é útil para a validação da arquitetura, permitindo uma revisão e avaliação crítica por parte dos stakeholders.

1.3 Definições, Acrônimos e Abreviações

- API: Sigla para "Interface de Programação de Aplicativos" (Application Programming Interface). É um conjunto de regras e protocolos que permite a interação entre diferentes componentes de software.
- MVC: É um padrão arquitetural amplamente utilizado em desenvolvimento de software para separar as responsabilidades em três componentes principais: Modelo (Model), Visão (View) e Controlador (Controller).
- Back-end: Refere-se à parte do sistema responsável pelo processamento e armazenamento de dados, regras de negócio e lógica de aplicação, geralmente executado no lado do servidor.
- Banco de Dados: Um sistema de armazenamento organizado de informações em formato eletrônico, projetado para permitir a recuperação, atualização e gerenciamento eficiente dos dados.
- Front-end: Refere-se à parte do sistema com a qual os usuários interagem diretamente. Geralmente é uma interface gráfica, como um aplicativo web ou mobile.
- Java Spring: Um framework de desenvolvimento de aplicativos Java baseado em Injeção de Dependência e Programação Orientada a Aspectos (AOP), que facilita o desenvolvimento de aplicativos corporativos robustos.
- PostgreSQL: Um sistema de gerenciamento de banco de dados relacional de código aberto e poderoso, conhecido por sua confiabilidade, escalabilidade e recursos avançados.
- React: Uma biblioteca JavaScript popular para construção de interfaces de usuário interativas e componentes reutilizáveis.

1.4 Referências

- Documento de Visão (DocumetodeVisãodoNegócioMocFrota).
- Documento de Casos de Uso (DocumentodeCasosdeUsoMocFrota).
- Guedes, Gilleanes T.A. UML: Uma abordagem prática. 2. ed. São Paulo: Novatec Editora, 2009.

1.5 Visão Geral

A partir desse ponto, para um melhor entendimento do documento, o mesmo foi dividido em 10 partes, as quais tratam de informações necessárias para o desenvolvimento do MocFrota. Cada tópico traz consigo suas características e importâncias para esse documento. Esses tópicos estão estruturados da seguinte forma:

2. Representação Arquitetural;
3. Metas e Restrições da Arquitetura;
4. Visão de Casos de Uso
5. Visão Lógica;
6. Visão de Processos;
7. Visão de Implantação;
8. Visão da Implementação;
9. Tamanho e Desempenho;
10. Qualidade

2. Representação Arquitetural

A arquitetura do sistema é baseada no padrão MVC (Model-View-Controller), que separa as responsabilidades de manipulação de dados, apresentação e lógica de negócios em três camadas: Camada de Apresentação, Camada de Aplicação e Camada de Dados. A arquitetura em camadas promove a modularidade, facilitando a manutenção, o teste e a evolução do sistema. Além disso, essa abordagem permite escalabilidade, uma vez que cada camada pode ser dimensionada independentemente das outras. A Camada de Apresentação é responsável por fornecer a interface do usuário e interagir com os usuários. A Camada de Aplicação é responsável por processar as solicitações dos usuários, aplicar as regras de negócios e coordenar as operações necessárias A Camada de Dados é responsável por interagir com o banco de dados e realizar operações de leitura e gravação de dados.

3. Metas e Restrições da Arquitetura

As principais metas são:

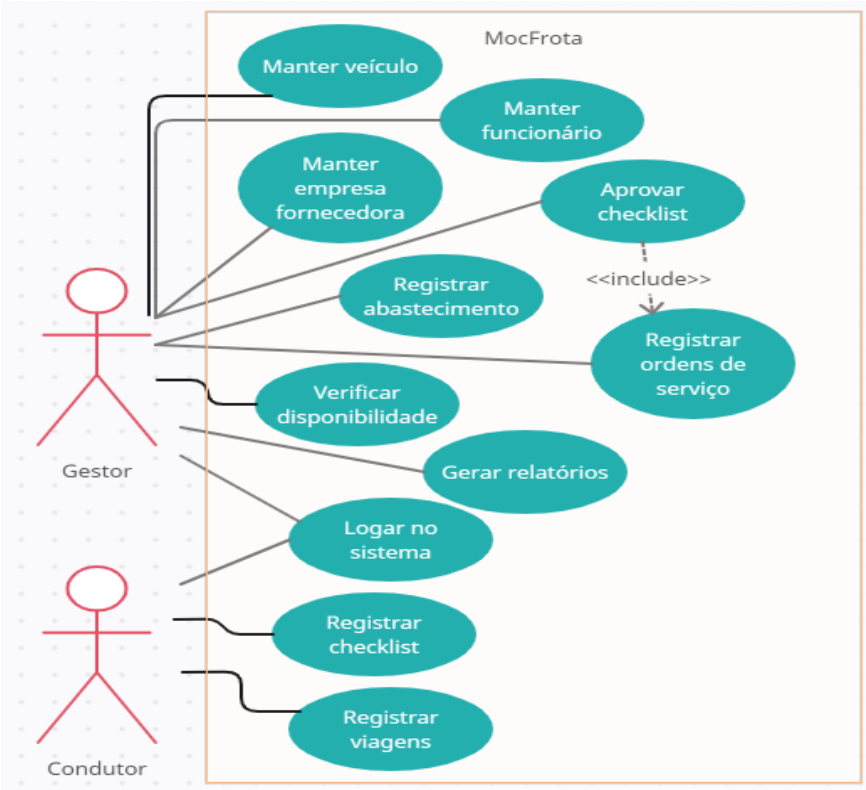
- Segurança: O sistema deve garantir a segurança dos dados e das transações.
- Desenvolvimento modular: A arquitetura deve permitir a modularidade e o baixo acoplamento entre os componentes do sistema, facilitando a manutenção, a escalabilidade e a extensibilidade.
- Usabilidade: O sistema deve ser intuitivo e fácil de usar, proporcionando uma experiência amigável para os usuários finais.
- Desempenho: O sistema deve ser capaz de lidar com um médio volume de dados.
- Escalabilidade: A arquitetura deve permitir que o sistema seja facilmente escalável verticalmente (adicionando recursos a um único servidor).

As principais restrições são:

- O sistema não conta com nenhuma restrição de hardware específica. Sendo utilizado as mesmas restrições de um navegador Web.

4. Visão de Casos de Uso

A figura abaixo representa todos os casos de uso que o sistema atende:



As especificações de todos os casos de uso estão descritas no Documento de Casos de Uso.

4.1 Realizações de Casos de Uso

Algumas realizações de Casos de Uso citados acima estão sendo ilustrada abaixo por meio de um Diagrama UML de sequência.

4.1.1 Logar no sistema:

4.1.2 Manter veículo:

4.1.3 Manter empresa:

4.1.4 Manter funcionário:

5. Visão Lógica

5.1 Visão Geral

A visão lógica descreve a estrutura interna e a organização lógica do sistema em termos de camadas e hierarquia de pacotes. Essa visão permite uma compreensão clara das partes significativas do sistema e sua divisão em subsistemas e pacotes.

5.2 Pacotes de Design Significativos do Ponto de Vista da Arquitetura

- Pacote "thigassantos.github.io.MocFrota": Este pacote contém as classes responsáveis por configurar e inicializar a aplicação.
    - Classe "MocFrotaApplication":
      Responsabilidades: Configuração e inicialização da aplicação Spring Boot.
      Relacionamentos: N/A.
      Operações e Atributos: N/A.
- Pacote "thigassantos.github.io.MocFrota.controller": Este pacote contém as classes controladoras que lidam com as requisições HTTP e gerenciam a interação entre o front-end e o back-end.
    - Classe "UserController":
      Responsabilidades: Lidar com as requisições relacionadas aos usuários, como registro e autenticação.
      Relacionamentos: Depende das classes de serviço relacionadas.
      Operações e Atributos: createUser(), authenticateUser(), updateUser(), etc.
- Pacote "thigassantos.github.io.MocFrota.repository": Este pacote contém as classes de repositório que lidam com o acesso e a persistência dos dados no banco de dados.
    - Classe "UserRepository":
      Responsabilidades: Realizar operações de acesso e persistência de dados relacionados aos usuários no banco de dados.
      Relacionamentos: N/A.
      Operações e Atributos: saveUser(), findUserById(), findUserByEmail(), etc.
- Pacote "thigassantos.github.io.MocFrota.model": Este pacote contém as classes de modelo que representam as entidades do sistema.
    - Classe "User":
      Responsabilidades: Representar um usuário no sistema.
      Relacionamentos: N/A.
      Operações e Atributos: id, name, email, password, etc.

6. Visão de Processos

Processos:

- Processo de Autenticação:
  Descrição: Este processo é responsável por autenticar os usuários e gerenciar as sessões de login.
  Comunicação: Os processos leves podem se comunicar usando chamadas de método dentro do mesmo processo, compartilhando informações sobre o status de autenticação e as sessões de usuário.
- Processo de Renderização de Páginas:
  Descrição: Este processo é responsável por renderizar as páginas do front-end do sistema.
  Comunicação: O processo de renderização pode se comunicar com o processo de gerenciamento de postagem para obter as informações necessárias para exibir as postagens na interface do usuário. Isso pode ser feito por meio de solicitações HTTP ou API de serviço.
- Processo de Atualização de Dados do Banco de Dados:
  Descrição: Este processo é responsável por atualizar os dados no banco de dados, garantindo a consistência e integridade dos dados.
  Comunicação: O processo de atualização de dados pode se comunicar com os processos de gerenciamento de postagem e autenticação para obter informações atualizadas para persistir no

banco de dados. Isso pode ser feito por meio de chamadas de método ou transações de banco de dados.

Modos de Comunicação:

- Transmissão de Mensagens: Os processos podem trocar mensagens contendo informações relevantes, como solicitações de ação ou atualizações de estado.
- Chamadas de Método: Os processos podem se comunicar por meio de chamadas de método entre os objetos ou componentes compartilhados dentro do mesmo processo.
- Requisições HTTP: Processos podem se comunicar usando protocolo HTTP, enviando solicitações e recebendo respostas por meio de APIs RESTful.

7. Visão de Implantação

- Servidor de Aplicativos:
  Descrição: Este nó físico é responsável por hospedar o servidor de aplicativos que executa o back-end do sistema.
  Processos Mapeados: Os processos de autenticação e atualização de dados do banco de dados são implantados neste servidor.
- Servidor Web:
  Descrição: Este nó físico é responsável por hospedar o servidor web que serve as páginas do front-end do sistema.
  Processos Mapeados: O processo de renderização de páginas é implantado neste servidor.
- Banco de Dados:
  Descrição: Este nó físico hospeda o banco de dados PostgreSQL que armazena os dados do sistema.
  Processos Mapeados: Os processos de atualização de dados do banco de dados se comunicam com este nó para persistir as informações.
- Interconexões:
  Conexão de Rede Local: Os nós físicos estão interconectados por meio de uma rede local (LAN), permitindo a comunicação e a troca de dados entre eles.

8. Visão da Implementação

8.1 Visão Geral

A visão da implementação descreve a estrutura geral do modelo de implementação, incluindo a divisão do software em camadas, subsistemas e componentes significativos do ponto de vista da arquitetura.

8.2 Camadas

- Camada de Apresentação (Front-end):
  Subsistemas: Interface do Usuário, Componentes de Interface, Gerenciamento de Estado
  Descrição: Esta camada lida com a apresentação visual e interação do sistema, fornecendo uma interface amigável para os usuários. Ela contém os subsistemas responsáveis pela criação e manipulação dos componentes da interface do usuário, bem como o gerenciamento do estado da aplicação.
- Camada de Aplicação (Back-end):
  Subsistemas: Controladores, Serviços, Repositórios
  Descrição: Esta camada lida com a lógica de negócio do sistema. Ela contém os subsistemas responsáveis pelo processamento das requisições recebidas, implementando as regras de negócio, usando e manipulando os dados do banco de dados.
- Camada de Persistência (Banco de Dados):
  Subsistemas: Banco de Dados (PostgreSQL)
  Descrição: Esta camada é responsável pelo armazenamento persistente dos dados do sistema. Ela contém o subsistema do banco de dados, que gerencia as tabelas, consultas e transações do banco de dados.

9. Tamanho e Desempenho

As medidas de desempenho que o MocFrota deve atender estão descritas abaixo, e estão relacionados a velocidade do sistema e o retorno para o usuário.

- Tempo de resposta (em uma variedade de condições de carregamento): O sistema deverá retornar o resultado em no máximo 2 segundos em condições normais de uso e carga.
- Capacidade: O sistema deverá ser capaz de lidar com até 20 usuários simultâneos, sem mostrar perda em sua velocidade e tempo de resposta em suas operações.
- Largura de banda ou capacidade de comunicação: O sistema deve ser capaz de lidar com uma taxa de transferência de no mínimo 10 Mbps.
- Rendimento: O sistema deve ser capaz de processar um volume médio de dados sem comprometer o desempenho.
- Exatidão: O sistema deve fornecer resultados precisos e confiáveis.
- Confiabilidade: O sistema deve ser altamente confiável e estar disponível em horário comercial já que o banco de dados é local.
- Outros: O tamanho e desempenho do sistema também dependerão das especificações do servidor web utilizado para a implantação.

10. Qualidade

A arquitetura de três camadas contribui para vários recursos do sistema, além da funcionalidade em si:

- Modularidade: A arquitetura de três camadas permite uma divisão clara do sistema em camadas distintas, cada uma com suas responsabilidades específicas. Isso promove a modularidade do sistema, facilitando a manutenção, extensibilidade e reutilização de componentes individuais.
- Manutenibilidade: Com a separação das camadas, a arquitetura facilita a manutenção do sistema. Cada camada pode ser desenvolvida, testada e mantida de forma independente, o que torna mais fácil identificar e corrigir problemas específicos em cada camada sem afetar as demais. Além disso, a modularidade proporcionada pela arquitetura facilita a adição de novos recursos e a implementação de melhorias no sistema.
- Desempenho e escalabilidade: A arquitetura permite a distribuição e escalabilidade dos componentes do sistema. As camadas podem ser implantadas em diferentes servidores ou máquinas, permitindo o balanceamento de carga e o aumento da capacidade de processamento. Isso contribui para um melhor desempenho e capacidade de lidar com um aumento na demanda.
- Segurança: A arquitetura de três camadas ajuda a melhorar a segurança do sistema. As camadas podem ser protegidas separadamente, permitindo a implementação de medidas de segurança específicas em cada camada.
- Testabilidade: A arquitetura de três camadas facilita a testabilidade do sistema. Como as camadas são separadas e têm responsabilidades específicas, é mais fácil isolar cada camada para testes unitários, de integração e de aceitação. Isso torna mais simples identificar e corrigir problemas em componentes individuais e garante a qualidade e confiabilidade do sistema como um todo.