**Documentação de Sistema**

Sistema de Gerenciamento de Logística (SGL)

Autores:

Luan Eron Neves Pinto – 799890

Pedro Henrique de Jesus - 896442

Belo Horizonte

02 de julho de 2025

**Sumário**

1. Introdução
   1. Escopo do Sistema
2. Nome do sistema e de seus componentes principais
3. Missão ou objetivo do sistema
   1. Arquitetura do Sistema
   2. Funcionalidade do sistema
   3. Usuários e sistemas externos
4. Documentação do código
   1. Documentação da Estrutura de dados geral do software
   2. Função adicionarLocal(Repositorio\* repo)
   3. Função removerLocal(Repositorio\* repo)
   4. Função salvarBinario(const char\* DB\_PATH)
   5. Função carregarBinario(const char\* DB\_PATH)
5. Testes de Software

**Introdução**

**Escopo do Sistema**

**Nome do sistema e de seus componentes principais**

O sistema é denominado **Sistema de Gerenciamento de Logística (SGL)**. Seus componentes principais são:

* **Entidades de Dados**: Classes que modelam os objetos do mundo real (Endereco, Local, Veiculo, Pedido).
* **Repositório**: Uma classe central que gerencia coleções de entidades de dados em memória, atuando como um banco de dados temporário.
* **Serviços (Services)**: Classes que contêm a lógica de negócio para manipular as entidades (ex: LocalService).
* **Verificadores**: Funções de validação para garantir a integridade dos dados antes de serem processados (ex: verificaLocal, verificaPedido).
* **Persistência**: Funções para salvar e carregar o estado do sistema em um arquivo binário.

**Missão ou objetivo do sistema**

A missão do sistema é gerenciar as operações de uma pequena empresa de logística, permitindo o cadastro, consulta, atualização e remoção de locais de entrega, veículos e pedidos. O sistema também deve ser capaz de salvar e carregar seus dados para garantir a persistência das informações entre as execuções.

**Arquitetura do Sistema**

O sistema possui uma arquitetura em camadas, projetada para separar as responsabilidades e facilitar a manutenção.

1. **Camada de Dados (Classes)**: A base do sistema, composta pelas classes Endereco, Local, Veiculo, e Pedido. Elas encapsulam os atributos e comportamentos fundamentais das entidades de negócio. Por exemplo, a classe Local é composta por um objeto Endereco , e a classe Pedido é composta por dois objetos Local (origem e destino).
2. **Camada de Persistência e Acesso a Dados (Repositório)**: A classe Repositorio atua como uma fachada para o acesso aos dados. Ela armazena vetores de locais, veiculos, e pedidos. Todas as operações de CRUD (Criar, Ler, Atualizar, Deletar) são centralizadas aqui. Esta camada também é responsável por serializar (salvar) e desserializar (carregar) os dados de/para um arquivo binário (database.bin).
3. **Camada de Lógica de Negócio (Serviços e Verificadores)**:
   * **Verificadores**: Um conjunto de funções (verificaEndereco, verificaLocal, verificaPedido, verificaPlaca ) que validam a integridade dos dados. Por exemplo, verificaPlaca garante que uma placa de veículo tenha o formato de 3 letras e 4 números.
   * **Serviços**: Classes como LocalService que orquestram as operações. Elas utilizam o Repositorio para acessar os dados e aplicam a lógica de negócio, como criar ou atualizar um local.
4. **Camada de Apresentação (main e funções de interface)**: A função main e funções auxiliares como adicionarLocal , listarLocais , e editarLocal interagem diretamente com o usuário através do console. Elas coletam os dados do usuário, invocam os serviços apropriados para processar as requisições e exibem os resultados.

O fluxo de dados é tipicamente iniciado na Camada de Apresentação, que passa as requisições para a Camada de Lógica de Negócio (Serviços). Os serviços, por sua vez, utilizam a Camada de Repositório para manipular os dados, que são então persistidos no disco.

**Funcionalidade do sistema**

|  |  |
| --- | --- |
| 1 | Adicionar um novo local com endereço e coordenadas. |
| 2 | Listar todos os locais cadastrados, exibindo ID, endereço e coordenadas. |
| 3 | Editar as informações de um local existente a partir de seu ID. |
| 4 | Remover um local existente do sistema a partir de seu ID. |
| 5 | Cadastrar, consultar, atualizar e remover veículos. |
| 6 | Cadastrar, consultar, atualizar e remover pedidos. |
| 7 | Salvar todas as informações (locais, veículos, pedidos) em um arquivo binário. |
| 8 | Carregar todas as informações a partir de um arquivo binário no início da execução. |

**Usuários e sistemas externos**

|  |  |  |
| --- | --- | --- |
|  | Usuários | Definição |
| 1 | Operador do Sistema | Um usuário que interage com o sistema através de um console de linha de comando para gerenciar os dados. |
| 2 | Sistema de Arquivos | O sistema interage com o sistema de arquivos local para salvar e carregar o banco de dados. |

### Documentação do código

#### Documentação da Estrutura de dados geral do software

A principal estrutura para armazenamento de dados em tempo de execução é a classe Repositorio. Ela utiliza três vetores da Biblioteca Padrão do C++ (std::vector) para armazenar os objetos principais do sistema:

* std::vector<Local> locais: Armazena todos os objetos do tipo Local cadastrados.
* std::vector<Veiculo> veiculos: Armazena todos os objetos do tipo Veiculo cadastrados.
* std::vector<Pedido> pedidos: Armazena todos os objetos do tipo Pedido cadastrados.

Essas estruturas de dados crescem dinamicamente conforme novos elementos são adicionados, o que é ideal para uma quantidade de dados que não é fixa. O acesso, a inserção e a remoção de elementos são gerenciados por métodos dentro da própria classe Repositorio.

#### Função adicionarLocal(Repositorio\* repo)

* **Assinatura**: void adicionarLocal(Repositorio\* repo)
* **Parâmetros**:
  + Repositorio\* repo: Ponteiro para o objeto Repositorio que armazena todos os dados do sistema.
* **Retorno**: void (nenhum).
* **Descrição**: Solicita ao usuário que digite os dados de um novo local (rua, cidade, estado e coordenadas) através do console. Em seguida, cria um objeto Local e o adiciona ao repositório usando a classe LocalService. Informa ao usuário se a criação foi bem-sucedida ou se ocorreu um erro.

#### Função removerLocal(Repositorio\* repo)

* **Assinatura**: void removerLocal(Repositorio\* repo)
* **Parâmetros**:
  + Repositorio\* repo: Ponteiro para o objeto Repositorio que armazena todos os dados do sistema.
* **Retorno**: void (nenhum).
* **Descrição**: Pede ao usuário o ID do local que deseja remover. Utiliza o LocalService para tentar deletar o local correspondente. Informa ao usuário se a remoção foi bem-sucedida ou se ocorreu um erro (ex: local não encontrado).

#### Função salvarBinario(const char\* DB\_PATH)

* **Assinatura**: void salvarBinario(const char\* DB\_PATH)
* **Parâmetros**:
  + const char\* DB\_PATH: Caminho do arquivo onde os dados serão salvos.
* **Retorno**: void (nenhum).
* **Descrição**: Abre um arquivo em modo de escrita binária. Escreve no arquivo o número de locais, veículos e pedidos, seguido pelos dados de cada objeto em sequência. Lança uma exceção std::runtime\_error se o arquivo não puder ser aberto.

#### Função carregarBinario(const char\* DB\_PATH)

* **Assinatura**: void carregarBinario(const char\* DB\_PATH)
* **Parâmetros**:
  + const char\* DB\_PATH: Caminho do arquivo de onde os dados serão lidos.
* **Retorno**: void (nenhum).
* **Descrição**: Abre um arquivo em modo de leitura binária. Lê o número de locais, veículos e pedidos e, em seguida, lê os dados de cada objeto, preenchendo os vetores correspondentes no Repositorio. Lança uma exceção std::runtime\_error se o arquivo não puder ser aberto.

**Testes de Software**