

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS E INFORMÁTICA
UNIDADE EDUCACIONAL PRAÇA DA LIBERDADE
Bacharelado em Engenharia de Software**

Nome dos integrantes do grupo:

Arthur Souza Fernandes

Victor Roberto Chagas Alves

Vinícius dos Santos Gonçalves

Nome do sistema:

**Sistema de Logística de Entrega
de Mercadorias (SLEM)**

Apresentação:

O objetivo deste projeto é desenvolver o Sistema de Logística de Entrega de Mercadorias (SLEM), uma aplicação em C++ para simular o gerenciamento de pedidos, veículos e locais de uma operação logística. O sistema permite o cadastro completo das entidades, calcula rotas de entrega baseadas no veículo mais próximo e garante a persistência dos dados entre sessões. O desenvolvimento foi guiado por metodologias ágeis, utilizando o GitHub Projects, calls no Discord e reuniões presenciais para organização do backlog, planejamento de sprints e acompanhamento do progresso das tarefas.

Backlog do produto:

No repositório do projeto no GitHub é possível acessar de forma completa na parte de "Projects" como foi o andamento e progresso sucessivo durante a produção do SLEM.

O projeto foi dividido em tarefas menores, atribuídas aos membros da equipe e organizadas em um quadro Kanban com as colunas: Backlog, A Fazer (To Do), Em Andamento (In Progress) e Concluído (Done). Além disso tinha mais duas colunas no quadro Kanban, mas as principais foram as citadas anteriormente.

O quadro principal no GitHub Projects foi estruturado para refletir o fluxo de trabalho do nosso time. A coluna "Backlog" continha todas as funcionalidades (Épicos) e tarefas a serem desenvolvidas, cada pessoa segundo sua disponibilidade - discutida durante as reuniões - ficou responsável por realizar alguns dos Épicos estabelecidos.

Sprint 1: Estrutura Base e Módulo de Locais

A primeira sprint focou em estabelecer a fundação do projeto e entregar o primeiro módulo funcional completo.

- **Objetivo da Sprint:** Criar a estrutura de dados, o menu principal e implementar o gerenciamento completo de Locais.
- **Tarefas:**
 - **Definir structs:** Criar as estruturas Local, Veiculo e Pedido no ficheiro `slem.h`.
 - **Implementar Menus:** Desenvolver as funções `exibirMenuPrincipal` e `exibirSubMenu`.
 - **CRUD de Locais:** Implementar as funções `cadastrarLocal`, `listarLocais`, `atualizarLocal`, `excluirLocal` e `buscarLocalPorNome`.
 - **Testes de Locais:** Criar o teste unitário para a função `buscarLocalPorNome`.

Sprint 2: Módulos de Veículos e Pedidos

Com a base pronta, a segunda sprint focou em adicionar os módulos de gerenciamento restantes.

- **Objetivo da Sprint:** Implementar o gerenciamento completo de Veículos e Pedidos.
- **Tarefas:**
 - **CRUD de Veículos:** Implementar as funções de cadastro, listagem, atualização e exclusão de veículos.
 - **Testes de Veículos:** Criar o teste unitário para a função `buscarVeiculoPorPlaca`.
 - **CRUD de Pedidos:** Implementar as funções de cadastro, listagem e exclusão de pedidos.

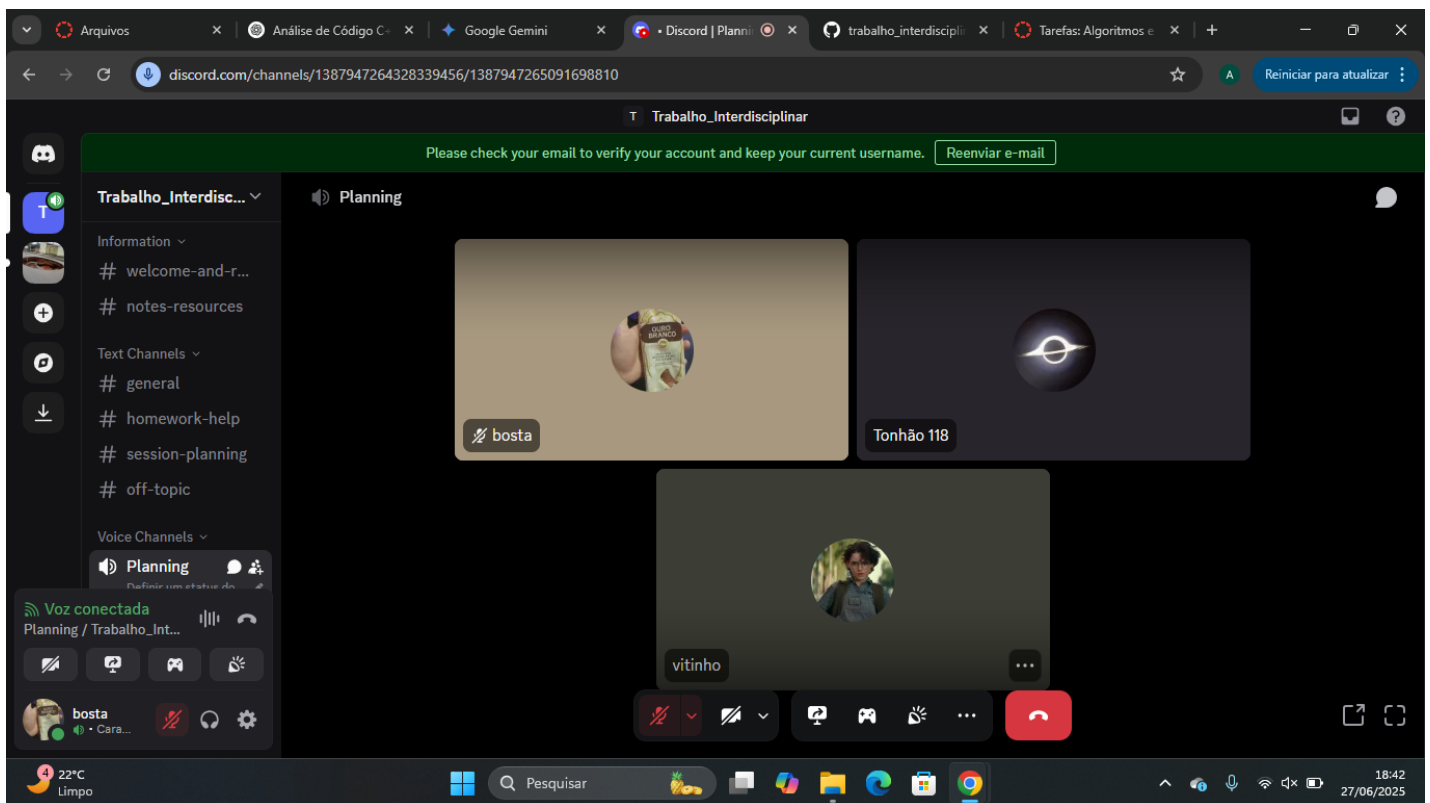
- **Testes de Pedidos:** Criar o teste unitário para a função `buscarPedidoPorId`.

Sprint 3: Roteamento e Persistência de Dados

A última sprint de desenvolvimento focou nas funcionalidades mais complexas do sistema.

- **Objetivo da Sprint:** Implementar o cálculo de rotas e as funções de salvar/carregar dados.
- **Tarefas:**
 - **Função de Distância:** Implementar `calcularDistancia`.
 - **Lógica de Rota:** Desenvolver a função `calcularExibirRota`, que encontra o veículo mais próximo e exibe o trajeto.
 - **Testes de Rota:** Criar o teste unitário para a função `calcularDistancia`.
 - **Persistência:** Implementar as funções `salvarDados` e `restaurarDados`.

No GitHub é melhor de se visualizar como foi desenvolvido a aplicação, portanto colocarei abaixo alguns prints de nossas reuniões e da parte de “Projects” do GitHub.



Arquivos

Microsoft Power...

stcrp em C++

Google Gemini

Tarefas: Algorit...

Sent Mail - arth...

Discord | P...

discord.com/channels/1387947264328339456/1387947265091698810

Reiniciar para atualizar

Trabalho_Interdisciplinar

Please check your email to verify your account and keep your current username. [Reenviar e-mail](#)

Trabalho_Interdisc...

welcome-anu...

NOVAS MENSAGENS NÃO LIDAS

notes-resources

Text Channels

general

homework-help

session-planning

off-topic

Voice Channels

Planning

Definir um status do ...




bosta

Voz conectada




Planning / Trabalho_Int...

bosta

Cara...



Procurando alguma coisa para fazer?
Comece um jogo ou lista de vídeos instantaneamente.




☐ Não me mostre isso novamente.

NOVOS TEMAS! Dê a sua cara ao Discord

Pré-visualizar um tema

Assine o Nitro e deixe o Discord do seu jeito com cores exclusivas.




Desbloquear com Nitro

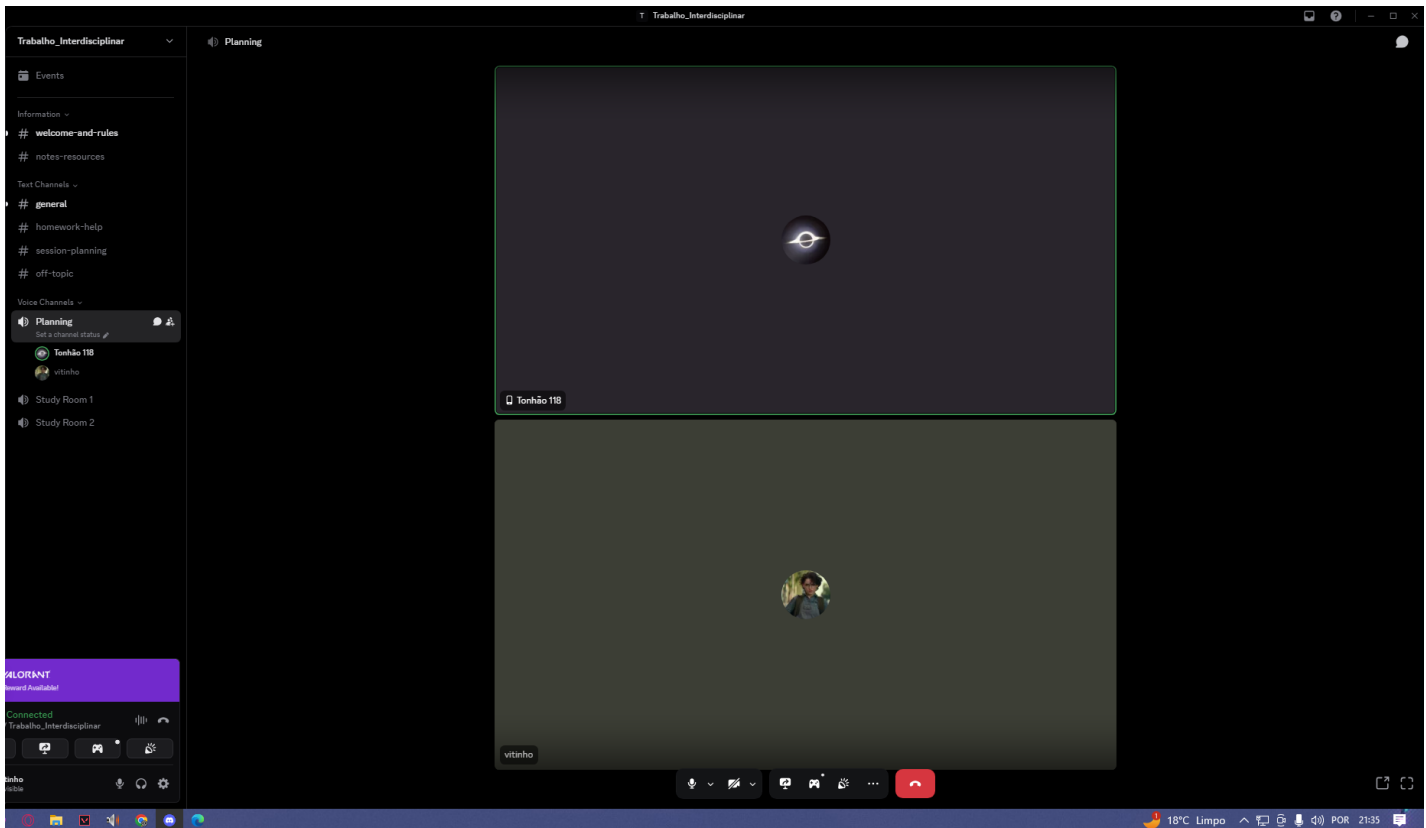
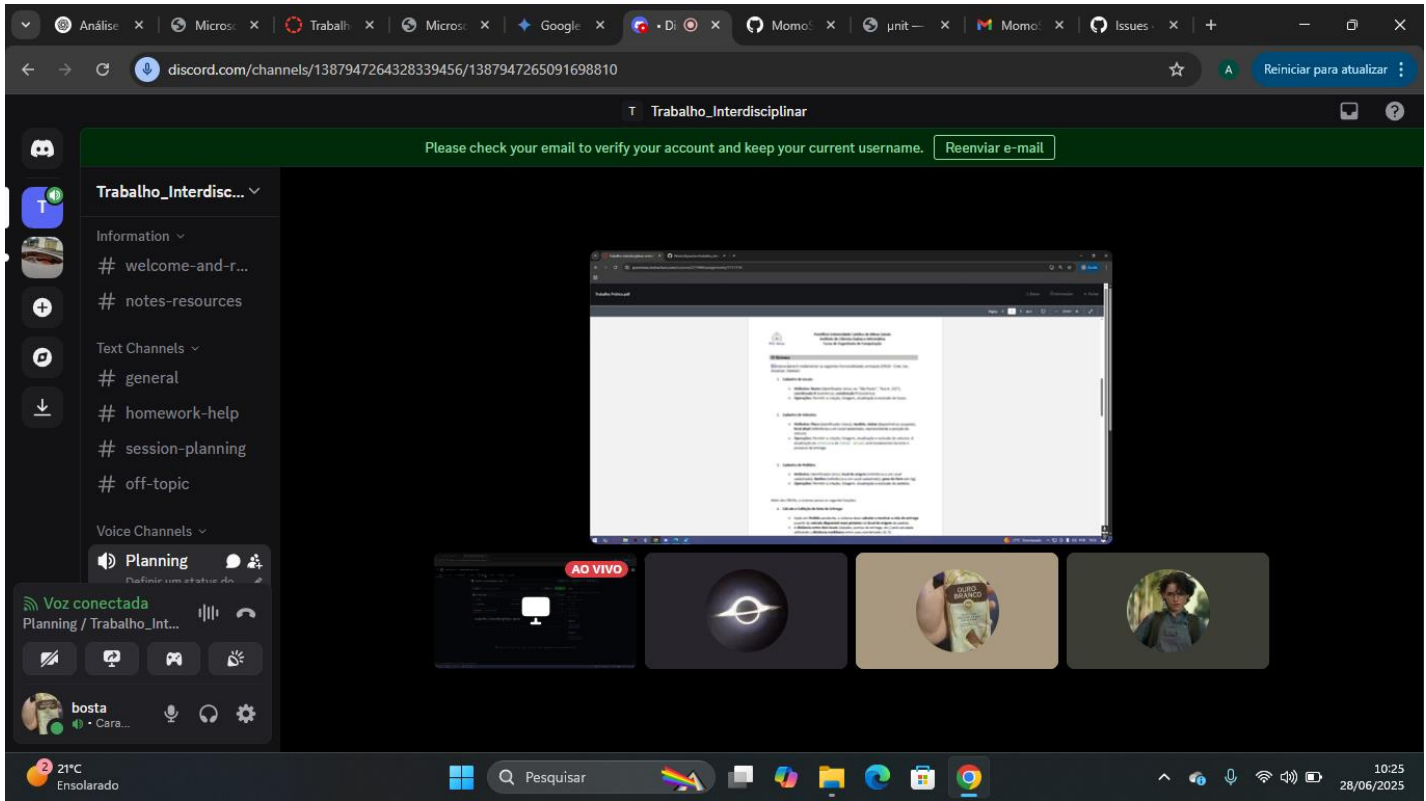
Fechar

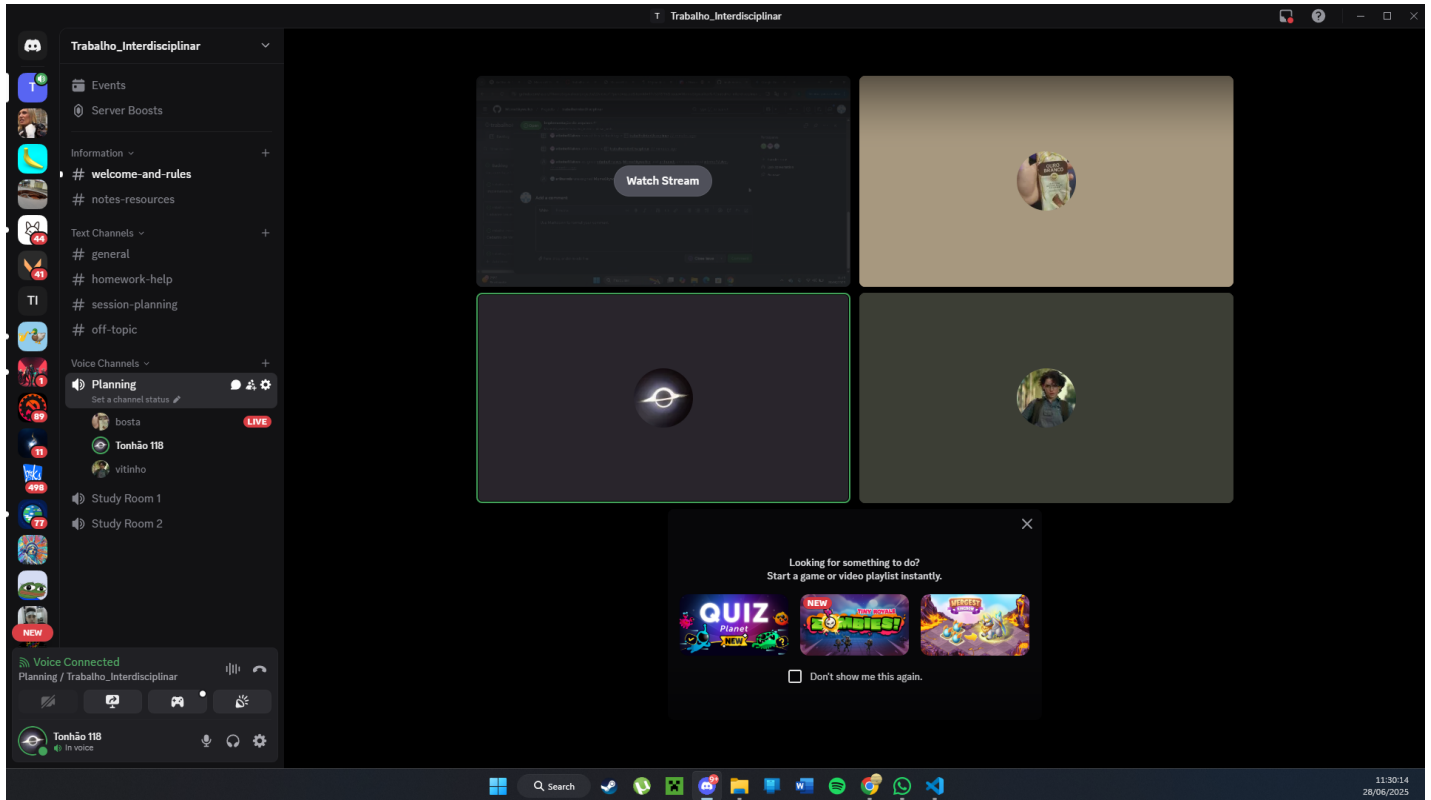
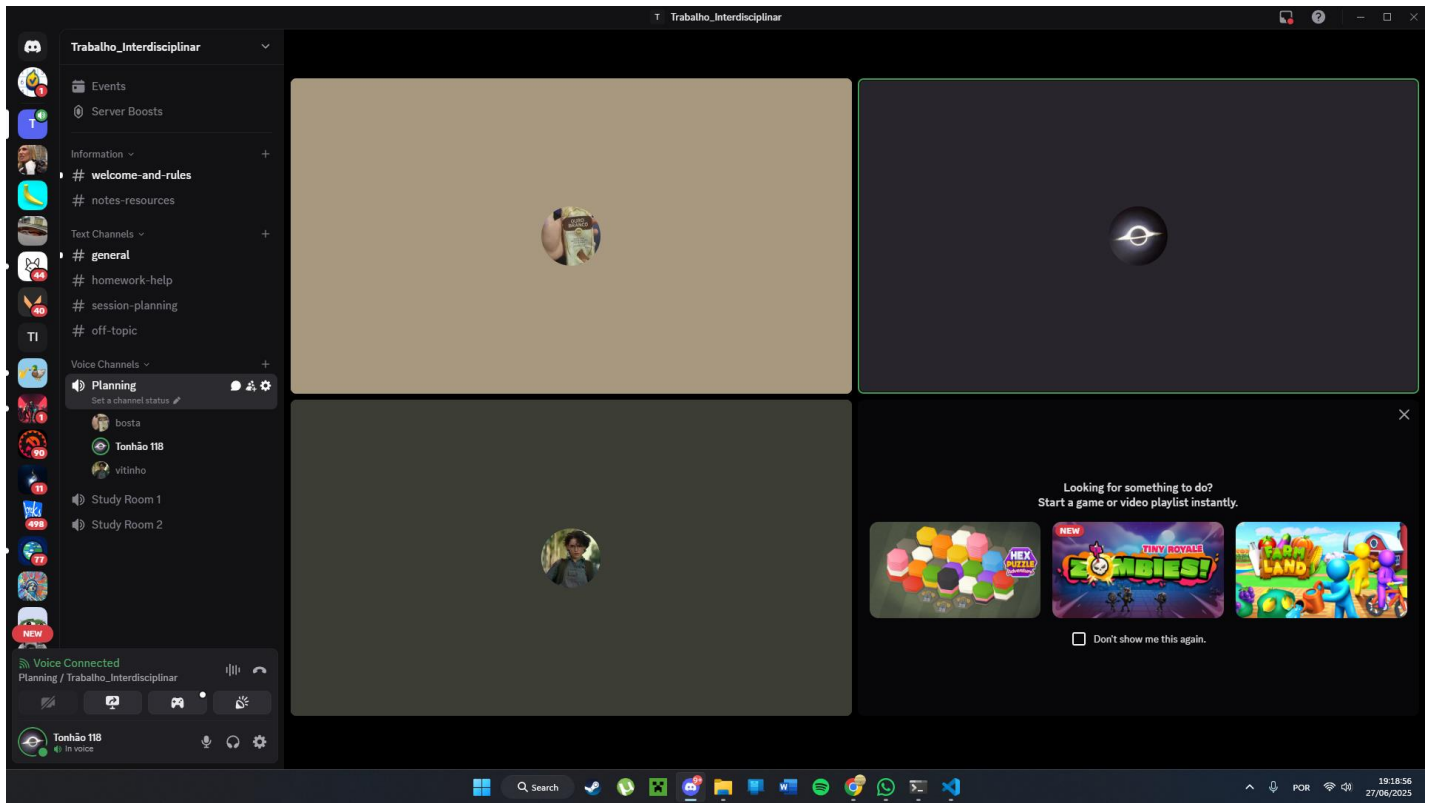
20°C Limpo

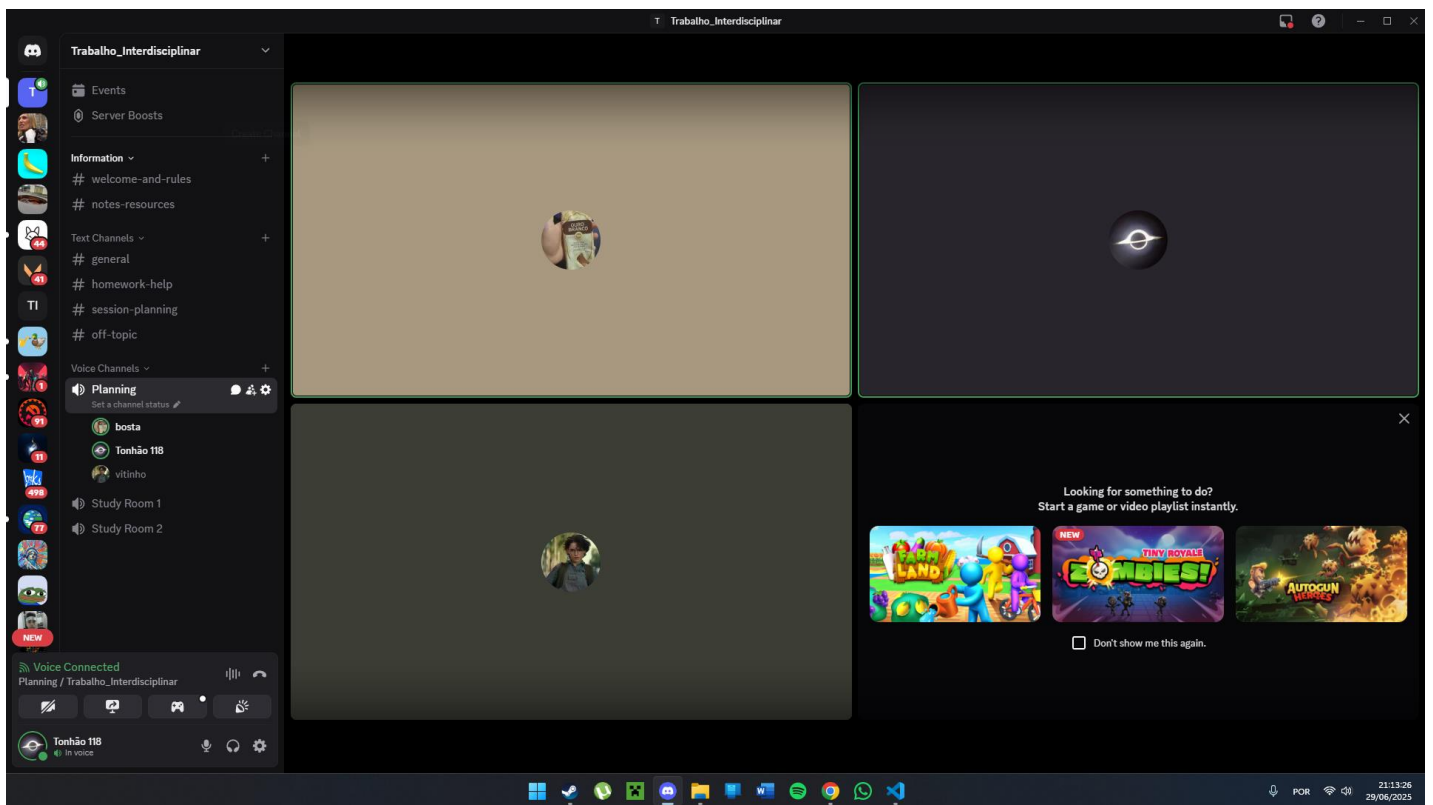
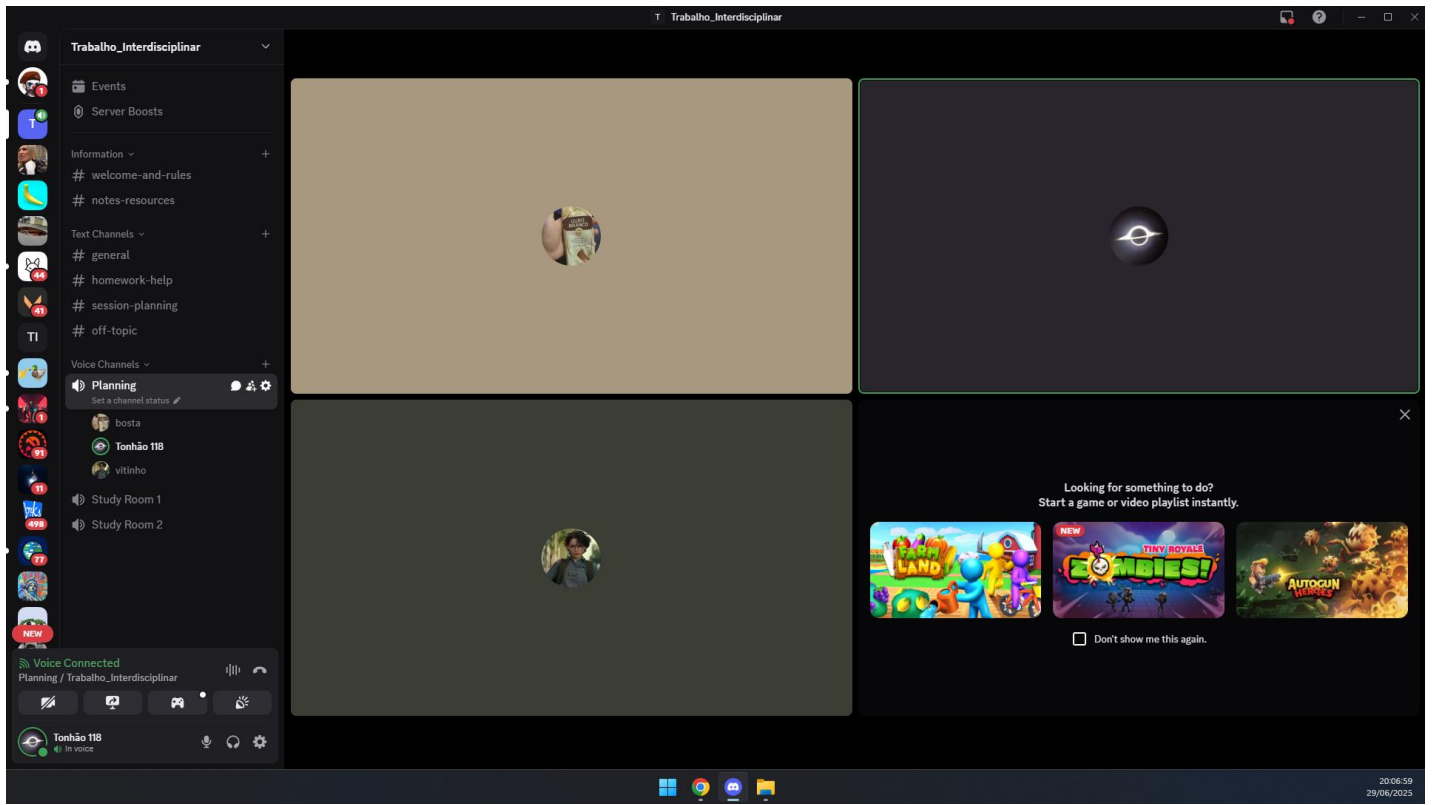
Pesquisar

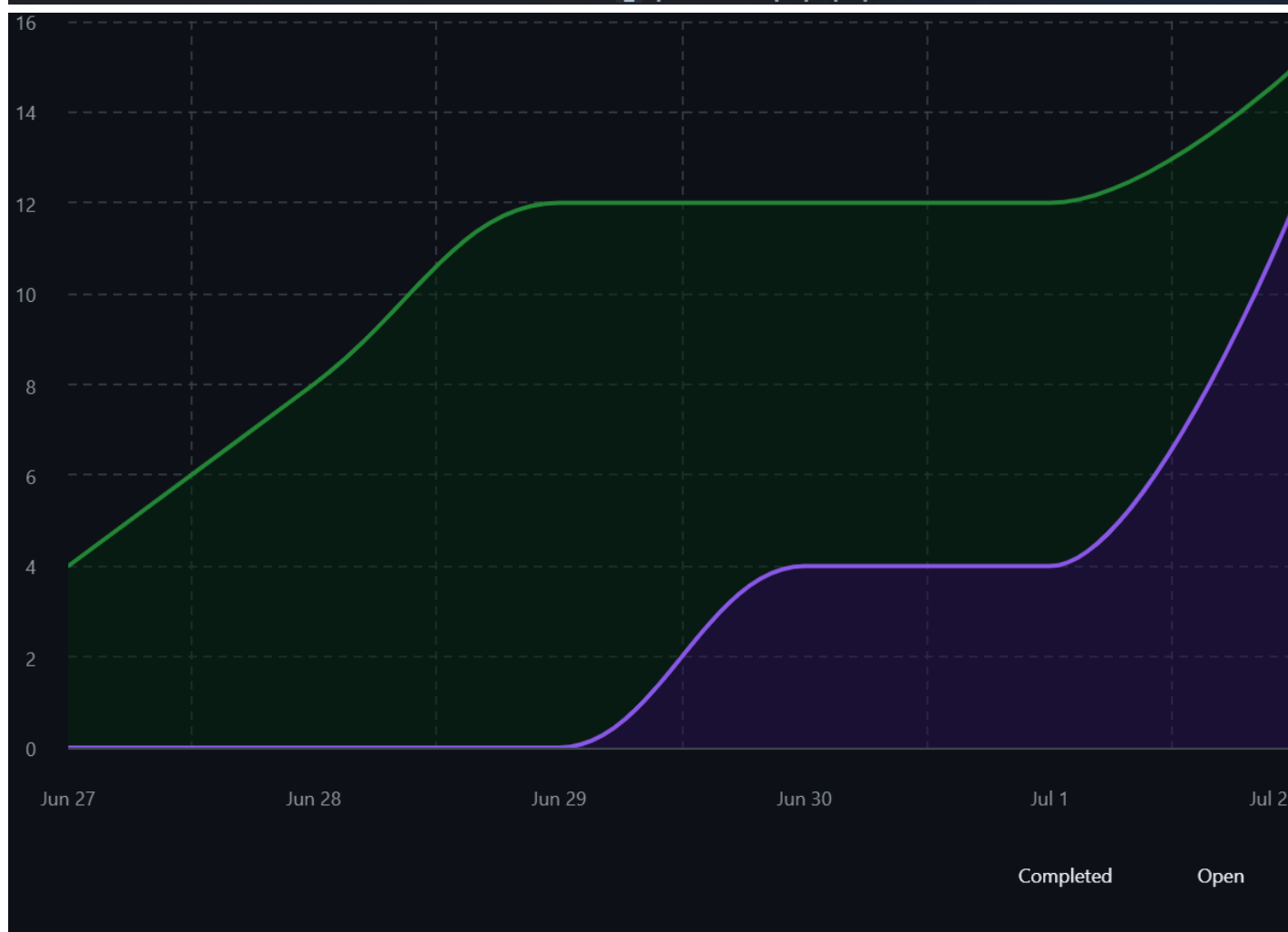
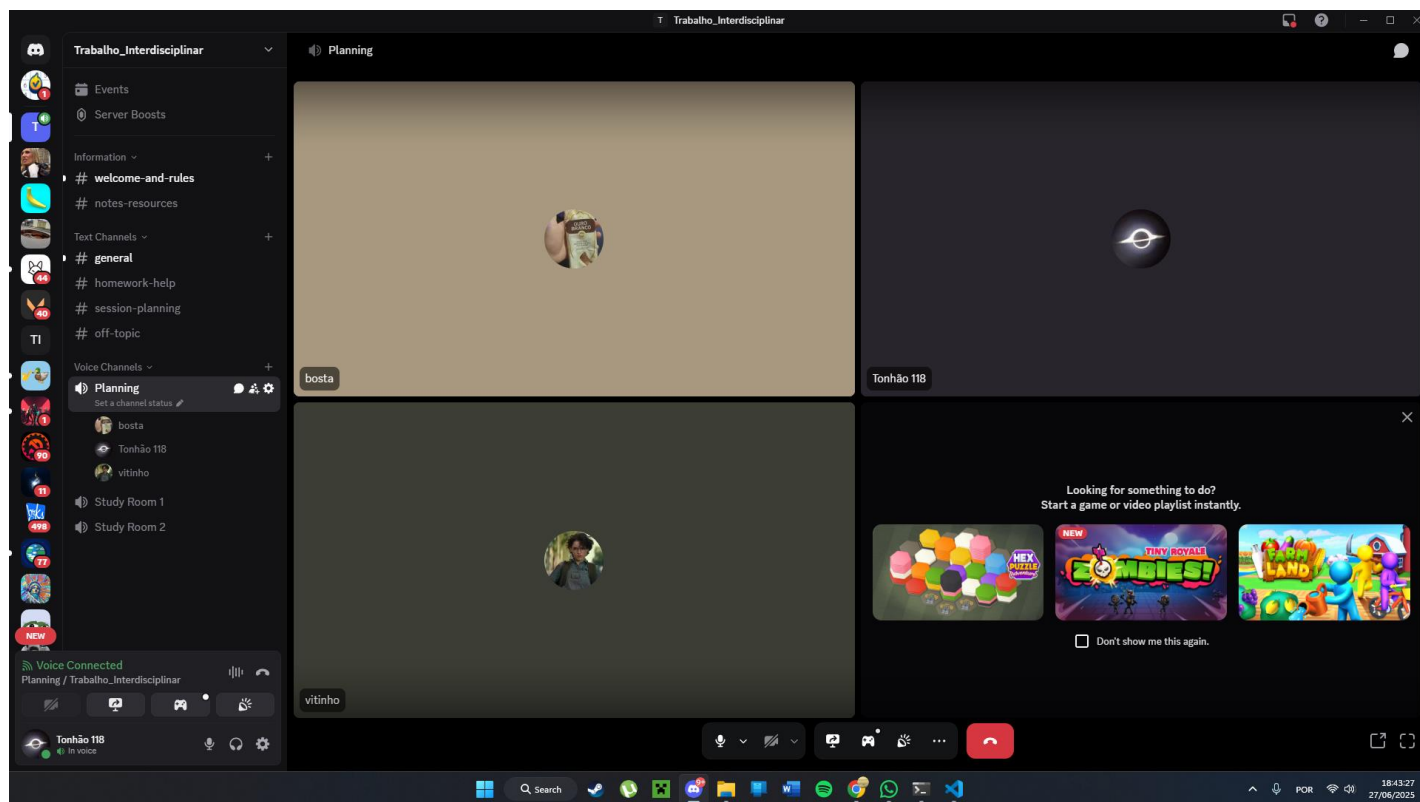


21:10
26/06/2025









Nota : As mudanças do Kanban ficam salvas no GitHub.

Lista de assinaturas das funções e parâmetros

Estrutura de Dados Principal

O sistema armazena todos os seus dados em memória principal através de vetores (arrays) de structs, definidas no ficheiro `slem.h`: Local, Veiculo e Pedido. Esta abordagem permite uma manipulação de dados estruturada e eficiente para as operações do sistema.

Assinaturas das Funções

1. `int buscarLocalPorNome (const char* nome, const Local locais[], int numLocais)`
 - a. Função para encontrar o índice de um local a partir do seu nome. Retorna o índice ou -1 se não for encontrado.
2. `int buscarVeiculoPorPlaca (const char* placa, const Veiculo veiculos[], int numVeiculos)`
 - a. Função para encontrar o índice de um veículo a partir da sua placa. Retorna o índice ou -1 se não for encontrado.
3. `int buscarPedidoPorId (int id, const Pedido pedidos[], int numPedidos)`
 - a. Função para encontrar o índice de um pedido a partir do seu ID. Retorna o índice ou -1 se não for encontrado.
4. `double calcularDistancia (const Local &l1, const Local &l2)`
 - a. Função para calcular a distância euclidiana entre dois locais. Retorna a distância como um double.
5. `void salvarDados(const Local[], int, const Veiculo[], int, const Pedido[], int)`
 - a. Função para salvar os dados atuais dos vetores de locais, veículos e pedidos em ficheiros binários. Não possui retorno.
6. `void restaurarDados(Local[], int&, Veiculo[], int&, Pedido[], int&)`
 - a. Função para carregar os dados dos ficheiros binários para os vetores em memória ao iniciar o programa. Não possui retorno.

TESTES

Os testes foram feitos de maneira automatizada. Além disso o programa também foi testado manualmente inúmeras vezes, se mostrando eficiente e de qualidade.

Segue os prints dos testes:

```
PS C:\Users\arthu\OneDrive\Desktop\trabalho_interdisciplinar_aeds> g++ testes/test_rota.cpp src/rota.cpp src/loais.cpp src/veiculos.cpp -o teste_rota
In file included from testes/test_rota.cpp:2:
testes/test_rota.cpp: In function 'char* test_calcularDistancia()':
testes/test_rota.cpp:11:15: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
   11 |     mu_assert("Erro: distancia entre (0,0) e (3,4) deve ser 5.0", (int)(dist * 10) == 50);
       |               ^~~~~~
testes/minunit.h:4:59: note: in definition of macro 'mu_assert'
    4 | #define mu_assert(message, test) do { if (!(test)) return message; } while (0)
       |                                                           ^~~~~~
PS C:\Users\arthu\OneDrive\Desktop\trabalho_interdisciplinar_aeds> ./teste_rota.exe
TODOS OS TESTES PASSARAM
Testes executados: 1
PS C:\Users\arthu\OneDrive\Desktop\trabalho_interdisciplinar_aeds>
```

```
PS C:\Users\arthu\OneDrive\Desktop\trabalho_interdisciplinar_aeds> g++ testes/test_pedidos.cpp src/pedidos.cpp src/loais.cpp -o teste_pedidos
In file included from testes/test_pedidos.cpp:2:
testes/test_pedidos.cpp: In function 'char* test_buscarPedidoPorId()':
testes/test_pedidos.cpp:12:15: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
   12 |     mu_assert("Erro: Pedido 1 deve estar na posicao 0", buscarPedidoPorId(1, pedidos, 2) == 0);
       |               ^~~~~~
testes/minunit.h:4:59: note: in definition of macro 'mu_assert'
    4 | #define mu_assert(message, test) do { if (!(test)) return message; } while (0)
       |                                                           ^~~~~~
testes/test_pedidos.cpp:13:15: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
   13 |     mu_assert("Erro: Pedido 2 deve estar na posicao 1", buscarPedidoPorId(2, pedidos, 2) == 1);
       |               ^~~~~~
testes/minunit.h:4:59: note: in definition of macro 'mu_assert'
    4 | #define mu_assert(message, test) do { if (!(test)) return message; } while (0)
       |                                                           ^~~~~~
testes/test_pedidos.cpp:14:15: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
   14 |     mu_assert("Erro: Pedido inexistente deve retornar -1", buscarPedidoPorId(3, pedidos, 2) == -1);
       |               ^~~~~~
testes/minunit.h:4:59: note: in definition of macro 'mu_assert'
    4 | #define mu_assert(message, test) do { if (!(test)) return message; } while (0)
       |                                                           ^~~~~~
PS C:\Users\arthu\OneDrive\Desktop\trabalho_interdisciplinar_aeds> ./teste_pedidos.exe
TODOS OS TESTES PASSARAM
Testes executados: 1
```

```
PS C:\Users\arthu\OneDrive\Desktop\trabalho_interdisciplinar_aeds> g++ testes/test_veiculos.cpp src/veiculos.cpp src/loais.cpp -o teste_veiculos
In file included from testes/test_veiculos.cpp:3:
testes/test_veiculos.cpp: In function 'char* test_buscarVeiculoPorPlaca()':
testes/test_veiculos.cpp:13:15: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
   13 |     mu_assert("Erro: ABC1234 deve estar na posicao 0", buscarVeiculoPorPlaca("ABC1234", veiculos, 2) == 0);
       |               ^~~~~~
testes/minunit.h:4:59: note: in definition of macro 'mu_assert'
    4 | #define mu_assert(message, test) do { if (!(test)) return message; } while (0)
       |                                                           ^~~~~~
testes/test_veiculos.cpp:14:15: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
   14 |     mu_assert("Erro: XYZ9876 deve estar na posicao 1", buscarVeiculoPorPlaca("XYZ9876", veiculos, 2) == 1);
       |               ^~~~~~
testes/minunit.h:4:59: note: in definition of macro 'mu_assert'
    4 | #define mu_assert(message, test) do { if (!(test)) return message; } while (0)
       |                                                           ^~~~~~
testes/test_veiculos.cpp:15:15: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
   15 |     mu_assert("Erro: Placa inexistente deve retornar -1", buscarVeiculoPorPlaca("ZZZ9999", veiculos, 2) == -1);
       |               ^~~~~~
testes/minunit.h:4:59: note: in definition of macro 'mu_assert'
    4 | #define mu_assert(message, test) do { if (!(test)) return message; } while (0)
       |                                                           ^~~~~~
PS C:\Users\arthu\OneDrive\Desktop\trabalho_interdisciplinar_aeds> ./teste_veiculos.exe
TODOS OS TESTES PASSARAM
Testes executados: 1
```

```

PS C:\Users\arthur\OneDrive\Desktop\trabalho_interdisciplinar_aeds> g++ testes/test_locais.cpp src/locais.cpp -o teste_locais
In file included from testes/test_locais.cpp:3:
testes/test_locais.cpp: In function 'char* test_buscarLocalPorNome()':
testes/test_locais.cpp:18:15: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
 18 |     mu_assert("Erro: Centro deve estar na posicao 0", buscarLocalPorNome("Centro", locais, 2) == 0);
    |               ^~~~~~
testes/minunit.h:4:59: note: in definition of macro 'mu_assert'
   4 | #define mu_assert(message, test) do { if (!(test)) return message; } while (0)
    |                                                           ^~~~~~
testes/test_locais.cpp:19:15: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
 19 |     mu_assert("Erro: Bairro deve estar na posicao 1", buscarLocalPorNome("Bairro", locais, 2) == 1);
    |               ^~~~~~
testes/minunit.h:4:59: note: in definition of macro 'mu_assert'
   4 | #define mu_assert(message, test) do { if (!(test)) return message; } while (0)
    |                                                           ^~~~~~
testes/test_locais.cpp:20:15: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
 20 |     mu_assert("Erro: Inexistente deve retornar -1", buscarLocalPorNome("Inexistente", locais, 2) == -1);
    |               ^~~~~~
testes/minunit.h:4:59: note: in definition of macro 'mu_assert'
   4 | #define mu_assert(message, test) do { if (!(test)) return message; } while (0)
    |                                                           ^~~~~~
PS C:\Users\arthur\OneDrive\Desktop\trabalho_interdisciplinar_aeds> ./teste_locais.exe
TODOS OS TESTES PASSARAM
Testes executados: 1

```

Testes de Comportamento: Funções salvarDados e restaurarDados

Número do Teste	Ação do Usuário	Comportamento Esperado do Sistema	Resultado Obtido	Aprova do ?
1	1. Iniciar o programa. 2. Cadastrar 2 locais. 3. Sair do programa (opção 0).	O programa deve invocar salvarDados. O ficheiro locais.dat deve ser criado/atualizado na pasta dados_bin/.	O ficheiro foi criado com sucesso.	Sim
2	1. Iniciar novamente o programa após o teste 5.1. 2. Listar os locais.	O programa invoca restaurarDados no início. A listagem deve mostrar os 2 locais cadastrados na sessão anterior.	Os dados foram carregados e listados corretamente.	Sim
3	1. Iniciar o programa sem que os ficheiros .dat existam. 2. Listar os locais.	O programa tenta restaurar os dados, mas não encontra os ficheiros. A listagem deve mostrar "Nenhum local cadastrado".	O sistema iniciou corretamente sem dados prévios.	Sim

Testes de Comportamento: Navegação de Menus

Número do Teste	Ação do Utilizador	Comportamento Esperado do Sistema	Resultado Obtido	Aprovado?
1	No menu principal, digitar a opção 1.	O sistema deve limpar a tela e exibir o submenu "Gerenciar Locais".	O submenu correto foi exibido.	Sim
2	No menu principal, digitar uma opção inválida (ex: 9).	O sistema deve exibir uma mensagem de "Opção inválida" e mostrar o menu principal novamente.	A mensagem de erro foi exibida e o menu recarregado.	Sim
3	No menu "Gerenciar Locais", digitar a opção 0.	O sistema deve retornar ao menu principal.	O programa retornou ao menu principal corretamente.	Sim
4	No menu principal, digitar um caractere não numérico (ex: a).	O sistema deve detectar a entrada inválida, solicitar um número e aguardar nova entrada sem encerrar a execução.	O sistema tratou o erro e pediu nova entrada.	Sim

O código C++ do programa e suas funções incluindo a implementação automatizada dos casos de testes:

Main.cpp:

```
#include <iostream>
#include <cstdlib> // Para system()
#include "slem.h"
#include "locais.h"
#include "veiculos.h"
#include "pedidos.h"
#include "rota.h"
#include "arquivos.h"

void exibirMenuPrincipal() {
    system("clear || cls");
    std::cout << "==== Sistema de Logistica de Entrega de Mercadorias (SLEM) ==== \n";
    std::cout << "1. Gerenciar Locais \n";
    std::cout << "2. Gerenciar Veiculos \n";
    std::cout << "3. Gerenciar Pedidos \n";
    std::cout << "4. Calcular e Exibir Rota de Entrega \n";
    std::cout << "5. Backup de Dados \n";
    std::cout << "6. Restaurar Dados \n";
    std::cout << "0. Sair \n";
    std::cout << "===== \n";
    std::cout << "Escolha uma opcao: ";
}

void exibirSubMenu(const char* titulo) {
    system("clear || cls");
    std::cout << "==== Gerenciar " << titulo << " ==== \n";
    std::cout << "1. Cadastrar \n";
    std::cout << "2. Listar \n";
    std::cout << "3. Atualizar \n";
    std::cout << "4. Excluir \n";
    std::cout << "0. Voltar ao Menu Principal \n";
    std::cout << "===== \n";
    std::cout << "Escolha uma opcao: ";
}
```

```

int main() {
    Local locais[MAX_ENTIDADES];
    Veiculo veiculos[MAX_ENTIDADES];
    Pedido pedidos[MAX_ENTIDADES];
    int numLocais = 0, numVeiculos = 0, numPedidos = 0;
    int opcao;

    restaurarDados(locais, numLocais, veiculos, numVeiculos, pedidos, numPedidos);

    do {
        exibirMenuPrincipal();
        std::cin >> opcao;
        std::cin.ignore();

        switch (opcao) {
            case 1: {
                int subOpcao;
                do {
                    exibirSubMenu("Locais");
                    std::cin >> subOpcao;
                    std::cin.ignore();
                    switch (subOpcao) {
                        case 1: cadastrarLocal(locais, numLocais); break;
                        case 2: listarLocais(locais, numLocais); break;
                        case 3: atualizarLocal(locais, numLocais); break;
                        case 4: excluirLocal(locais, numLocais); break;
                    }
                    if (subOpcao != 0 && subOpcao <= 4) { std::cout << "\nPressione ENTER
para continuar..."; std::cin.get(); }
                } while (subOpcao != 0);
                break;
            }
            case 2: {
                int subOpcao;
                do {
                    exibirSubMenu("Veiculos");
                    std::cin >> subOpcao;
                    std::cin.ignore();
                    switch (subOpcao) {
                        case 1: cadastrarVeiculo(veiculos, numVeiculos, locais, numLocais);
break;

                        case 2: listarVeiculos(veiculos, numVeiculos); break;
                        case 3: atualizarVeiculo(veiculos, numVeiculos, locais, numLocais);
break;

                        case 4: excluirVeiculo(veiculos, numVeiculos); break;
                    }
                    if (subOpcao != 0 && subOpcao <= 4) { std::cout << "\nPressione ENTER
para continuar..."; std::cin.get(); }
                } while (subOpcao != 0);
                break;
            }
            case 3: {
                int subOpcao;

```

```

do {
    exibirSubMenu("Pedidos");
    std::cin >> subOpcao;
    std::cin.ignore();
    switch (subOpcao) {
        case 1: cadastrarPedido(pedidos, numPedidos, locais, numLocais);
break;

        case 2: listarPedidos(pedidos, numPedidos); break;
        case 3: std::cout << "\nOpcao de atualizar pedido nao
disponivel.\n"; break;
        case 4: excluirPedido(pedidos, numPedidos); break;
    }
    if (subOpcao != 0 && subOpcao <= 4) { std::cout << "\nPressione ENTER
para continuar..."; std::cin.get(); }
    } while (subOpcao != 0);
    break;
}
case 4: {
    if (numPedidos == 0) {
        std::cout << "\nNenhum pedido para calcular rota.\n";
    } else if (numVeiculos == 0) {
        std::cout << "\nNenhum veiculo disponivel para realizar entrega.\n";
    } else {
        int idPedido;
        std::cout << "\nDigite o ID do pedido para calcular a rota: ";
        std::cin >> idPedido;
        std::cin.ignore();
        int indicePedido = buscarPedidoPorId(idPedido, pedidos, numPedidos);
        if (indicePedido != -1) {
            calcularExibirRota(pedidos[indicePedido], veiculos, numVeiculos,
locais, numLocais);
        } else {
            std::cout << "\nPedido com ID " << idPedido << " nao encontrado.\n";
        }
    }
    std::cout << "\nPressione ENTER para continuar...";
    std::cin.get();
    break;
}
case 5:
    salvarDados(locais, numLocais, veiculos, numVeiculos, pedidos, numPedidos);
    std::cout << "\nPressione ENTER para continuar...";
    std::cin.get();
    break;
case 6:
    restaurarDados(locais, numLocais, veiculos, numVeiculos, pedidos,
numPedidos);

    std::cout << "\nPressione ENTER para continuar...";
    std::cin.get();
    break;
case 0:
    salvarDados(locais, numLocais, veiculos, numVeiculos, pedidos, numPedidos);
    std::cout << "\nSaindo do sistema...\n";
    break;

```

```

        default:
            std::cout << "\nOpcao invalida. Tente novamente.\n";
            std::cout << "\nPressione ENTER para continuar...";
            std::cin.get();
        }
    } while (opcao != 0);

    return 0;
}

```

Arquivos.cpp:

```

#include <iostream>
#include <fstream>
#include "arquivos.h"

void salvarDados(const Local locais[], int numLocais, const Veiculo veiculos[], int
numVeiculos, const Pedido pedidos[], int numPedidos) {
    std::ofstream arqLoc("dados_bin/locais.dat", std::ios::binary);
    if (arqLoc.is_open()) {
        arqLoc.write((char*)&numLocais, sizeof(int));
        arqLoc.write((char*)locais, sizeof(Local) * numLocais);
        arqLoc.close();
    }

    std::ofstream arqVei("dados_bin/veiculos.dat", std::ios::binary);
    if (arqVei.is_open()) {
        arqVei.write((char*)&numVeiculos, sizeof(int));
        arqVei.write((char*)veiculos, sizeof(Veiculo) * numVeiculos);
        arqVei.close();
    }

    std::ofstream arqPed("dados_bin/pedidos.dat", std::ios::binary);
    if (arqPed.is_open()) {
        arqPed.write((char*)&numPedidos, sizeof(int));
        arqPed.write((char*)pedidos, sizeof(Pedido) * numPedidos);
        arqPed.close();
    }

    std::cout << "\nDados salvos com sucesso!\n";
}

void restaurarDados(Local locais[], int &numLocais, Veiculo veiculos[], int &numVeiculos,
Pedido pedidos[], int &numPedidos) {
    std::ifstream arqLoc("dados_bin/locais.dat", std::ios::binary);
    if (arqLoc.is_open()) {
        arqLoc.read((char*)&numLocais, sizeof(int));
        arqLoc.read((char*)locais, sizeof(Local) * numLocais);
        arqLoc.close();
    }

    std::ifstream arqVei("dados_bin/veiculos.dat", std::ios::binary);
    if (arqVei.is_open()) {
        arqVei.read((char*)&numVeiculos, sizeof(int));
    }
}

```

```

        arqVei.read((char*)veiculos, sizeof(Veiculo) * numVeiculos);
        arqVei.close();
    }

    std::ifstream arqPed("dados_bin/pedidos.dat", std::ios::binary);
    if (arqPed.is_open()) {
        arqPed.read((char*)&numPedidos, sizeof(int));
        arqPed.read((char*)pedidos, sizeof(Pedido) * numPedidos);
        arqPed.close();
    }

    std::cout << "\nDados restaurados com sucesso!\n";
}

```

Arquivos.h:

```

#ifndef ARQUIVOS_H
#define ARQUIVOS_H

#include "slem.h"

// Declarações das funções que estão em arquivos.cpp
void salvarDados(const Local locais[], int numLocais, const Veiculo veiculos[], int numVeiculos, const Pedido pedidos[], int numPedidos);
void restaurarDados(Local locais[], int &numLocais, Veiculo veiculos[], int &numVeiculos, Pedido pedidos[], int &numPedidos);

#endif

```

Locais.cpp:

```

#include <iostream>
#include <cstring>
#include "locais.h"

int buscarLocalPorNome(const char* nome, const Local locais[], int numLocais) {
    for (int i = 0; i < numLocais; ++i) {
        if (strcmp(locais[i].nome, nome) == 0) return i;
    }
    return -1;
}

void cadastrarLocal(Local locais[], int &numLocais) {
    if (numLocais >= MAX_ENTIDADES) {
        std::cout << "\nErro: limite maximo de locais atingido.\n";
        return;
    }
    Local novoLocal;
    std::cout << "Nome do local: ";
    std::cin.getline(novoLocal.nome, MAX_NOME);
    if (buscarLocalPorNome(novoLocal.nome, locais, numLocais) != -1) {
        std::cout << "\nErro: local ja existente.\n";
        return;
    }
    std::cout << "Coordenada X: ";
    std::cin >> novoLocal.coordX;
}

```



```

    std::cout << "Coordenada Y: ";
    std::cin >> novoLocal.coordY;
    std::cin.ignore();
    locais[numLocais++] = novoLocal;
    std::cout << "\nLocal cadastrado com sucesso.\n";
}

void listarLocais(const Local locais[], int numLocais) {
    if (numLocais == 0) {
        std::cout << "Nenhum local cadastrado.\n";
        return;
    }
    for (int i = 0; i < numLocais; ++i) {
        std::cout << "Nome: " << locais[i].nome << " | Coordenadas: (" << locais[i].coordX
        << ", " << locais[i].coordY << ")\n";
    }
}

void atualizarLocal(Local locais[], int numLocais) {
    char nomeBusca[MAX_NOME];
    std::cout << "Digite o nome do local: ";
    std::cin.getline(nomeBusca, MAX_NOME);
    int indice = buscarLocalPorNome(nomeBusca, locais, numLocais);
    if (indice == -1) {
        std::cout << "Local nao encontrado.\n";
        return;
    }
    std::cout << "Nova coordenada X: ";
    std::cin >> locais[indice].coordX;
    std::cout << "Nova coordenada Y: ";
    std::cin >> locais[indice].coordY;
    std::cin.ignore();
    std::cout << "Local atualizado com sucesso.\n";
}

void excluirLocal(Local locais[], int &numLocais) {
    char nomeBusca[MAX_NOME];
    std::cout << "Digite o nome do local a excluir: ";
    std::cin.getline(nomeBusca, MAX_NOME);
    int indice = buscarLocalPorNome(nomeBusca, locais, numLocais);
    if (indice == -1) {
        std::cout << "Local nao encontrado.\n";
        return;
    }
    for (int i = indice; i < numLocais - 1; ++i) {
        locais[i] = locais[i + 1];
    }
    numLocais--;
    std::cout << "Local excluido.\n";
}

```

Locais.h:

```

#ifndef LOCAIS_H
#define LOCAIS_H

```

```

#include "slem.h"

// Declarações das funções que estão em locais.cpp
void cadastrarLocal(Local locais[], int &numLocais);
void listarLocais(const Local locais[], int numLocais);
void atualizarLocal(Local locais[], int numLocais);
void excluirLocal(Local locais[], int &numLocais);
int buscarLocalPorNome(const char* nome, const Local locais[], int numLocais);

#endif

```

Pedidos.cpp:

```

#include <iostream>
#include <cstring>
#include "pedidos.h"
#include "locais.h"

int buscarPedidoPorId(int id, const Pedido pedidos[], int numPedidos) {
    for (int i = 0; i < numPedidos; ++i) {
        if (pedidos[i].id == id) return i;
    }
    return -1;
}

void listarPedidos(const Pedido pedidos[], int numPedidos) {
    if (numPedidos == 0) {
        std::cout << "Nenhum pedido cadastrado.\n";
        return;
    }
    for (int i = 0; i < numPedidos; ++i) {
        std::cout << "ID: " << pedidos[i].id
                    << " | Origem: " << pedidos[i].localOrigem
                    << " | Destino: " << pedidos[i].localDestino
                    << " | Peso: " << pedidos[i].peso << "kg\n";
    }
}

void cadastrarPedido(Pedido pedidos[], int &numPedidos, const Local locais[], int numLocais)
{
    if (numPedidos >= MAX_ENTIDADES) {
        std::cout << "\nErro: Limite maximo de pedidos atingido.\n";
        return;
    }
    if (numLocais < 2) {
        std::cout << "\nErro: E necessario ter pelo menos 2 locais cadastrados.\n";
        return;
    }
    Pedido novoPedido;
    novoPedido.id = numPedidos + 1;
    std::cout << "--- Cadastro de Novo Pedido ---\n";
    std::cout << "Local de Origem: ";
    std::cin.getline(novoPedido.localOrigem, MAX_NOME);
    if (buscarLocalPorNome(novoPedido.localOrigem, locais, numLocais) == -1) {

```

```

        std::cout << "Local de origem nao encontrado.\n";
        return;
    }
    std::cout << "Local de Destino: ";
    std::cin.getline(novoPedido.localDestino, MAX_NOME);
    if (buscarLocalPorNome(novoPedido.localDestino, locais, numLocais) == -1) {
        std::cout << "Local de destino nao encontrado.\n";
        return;
    }
    std::cout << "Peso do pedido (kg): ";
    std::cin >> novoPedido.peso;
    std::cin.ignore();
    pedidos[numPedidos++] = novoPedido;
    std::cout << "Pedido cadastrado com sucesso!\n";
}

void excluirPedido(Pedido pedidos[], int &numPedidos) {
    int id;
    std::cout << "Digite o ID do pedido a excluir: ";
    std::cin >> id;
    std::cin.ignore();
    int idx = buscarPedidoPorId(id, pedidos, numPedidos);
    if (idx == -1) {
        std::cout << "Pedido nao encontrado.\n";
        return;
    }
    for (int i = idx; i < numPedidos - 1; ++i) {
        pedidos[i] = pedidos[i + 1];
    }
    numPedidos--;
    std::cout << "Pedido excluido.\n";
}

```

Pedidos.h:

```

#ifndef PEDIDOS_H
#define PEDIDOS_H

#include "slem.h"

// Declarações das funções que estão em pedidos.cpp
void cadastrarPedido(Pedido pedidos[], int &numPedidos, const Local locais[], int numLocais);
void listarPedidos(const Pedido pedidos[], int numPedidos);
void excluirPedido(Pedido pedidos[], int &numPedidos);
int buscarPedidoPorId(int id, const Pedido pedidos[], int numPedidos);

#endif

```

Rota.cpp:

```

#include <iostream>
#include <cmath>
#include <cstdio>
#include <cstring>
#include "rota.h"

```

```

#include "locais.h"
#include "veiculos.h"

double calcularDistancia(const Local &l1, const Local &l2) {
    return sqrt(pow(l2.coordX - l1.coordX, 2) + pow(l2.coordY - l1.coordY, 2));
}

void calcularExibirRota(Pedido &pedido, Veiculo veiculos[], int numVeiculos, const Local
locais[], int numLocais) {
    int idxOrigem = buscarLocalPorNome(pedido.localOrigem, locais, numLocais);
    int idxDestino = buscarLocalPorNome(pedido.localDestino, locais, numLocais);

    if (idxOrigem == -1 || idxDestino == -1) {
        std::cout << "Erro: Local de origem ou destino nao encontrado.\n";
        return;
    }

    double menorDistancia = DBL_MAX;
    int idxVeiculo = -1;

    for (int i = 0; i < numVeiculos; ++i) {
        if (veiculos[i].status == DISPONIVEL) {
            int idxLocalVeiculo = buscarLocalPorNome(veiculos[i].localAtual, locais,
numLocais);
            if (idxLocalVeiculo != -1) {
                double dist = calcularDistancia(locais[idxOrigem], locais[idxLocalVeiculo]);
                if (dist < menorDistancia) {
                    menorDistancia = dist;
                    idxVeiculo = i;
                }
            }
        }
    }

    if (idxVeiculo == -1) {
        std::cout << "Nao ha veiculos disponiveis.\n";
        return;
    }

    double distTotal = menorDistancia + calcularDistancia(locais[idxOrigem],
locais[idxDestino]);

    std::cout << "\n--- Rota de Entrega ---\n";
    std::cout << "Pedido ID: " << pedido.id << "\n";
    std::cout << "Veiculo: " << veiculos[idxVeiculo].placa << "\n";
    std::cout << "Rota: " << veiculos[idxVeiculo].localAtual << " -> "
        << pedido.localOrigem << " -> " << pedido.localDestino << "\n";
    std::cout << "Distancia total: " << distTotal << "\n";

    veiculos[idxVeiculo].status = OCUPADO;
    // Em um sistema real, o status só voltaria para DISPONIVEL após a entrega.
    // Para simplificar, vamos simular que a entrega é feita e o veículo já volta a ficar
disponível.
    strcpy(veiculos[idxVeiculo].localAtual, pedido.localDestino);

```

```

    veiculos[idxVeiculo].status = DISPONIVEL;
    std::cout << "Entrega finalizada. Veiculo agora esta em: " <<
veiculos[idxVeiculo].localAtual << "\n";
}

```

Rota.h:

```

#ifndef ROTA_H
#define ROTA_H

#include "slem.h"

// Declarações das funções que estão em rota.cpp
double calcularDistancia(const Local &l1, const Local &l2);
void calcularExibirRota(Pedido &pedido, Veiculo veiculos[], int numVeiculos, const Local
locais[], int numLocais);

#endif

```

Slem.h:

```

#ifndef SLEM_H
#define SLEM_H

// Constantes globais para o projeto
const int MAX_ENTIDADES = 100;
const int MAX_NOME = 50;

// Enum para o status do veículo
enum StatusVeiculo { DISPONIVEL, OCUPADO };

// Definição da struct para um Local
struct Local {
    char nome[MAX_NOME];
    int coordX;
    int coordY;
};

// Definição da struct para um Veículo
struct Veiculo {
    char placa[10];
    char modelo[MAX_NOME];
    StatusVeiculo status;
    char localAtual[MAX_NOME];
};

// Definição da struct para um Pedido
struct Pedido {
    int id;
    char localOrigem[MAX_NOME];
    char localDestino[MAX_NOME];
    float peso;
};

#endif

```

Veiculos.cpp:

```
#include <iostream>
#include <cstring>
#include "veiculos.h"
#include "locais.h"

int buscarVeiculoPorPlaca(const char* placa, const Veiculo veiculos[], int numVeiculos) {
    for (int i = 0; i < numVeiculos; ++i) {
        if (strcmp(veiculos[i].placa, placa) == 0) return i;
    }
    return -1;
}

void listarVeiculos(const Veiculo veiculos[], int numVeiculos) {
    if (numVeiculos == 0) {
        std::cout << "Nenhum veiculo cadastrado.\n";
        return;
    }
    for (int i = 0; i < numVeiculos; ++i) {
        std::cout << "Placa: " << veiculos[i].placa
            << " | Modelo: " << veiculos[i].modelo
            << " | Local: " << veiculos[i].localAtual
            << " | Status: " << (veiculos[i].status == DISPONIVEL ? "Disponivel" :
"Ocupado")
            << "\n";
    }
}

void cadastrarVeiculo(Veiculo veiculos[], int &numVeiculos, const Local locais[], int
numLocais) {
    if (numVeiculos >= MAX_ENTIDADES) {
        std::cout << "\nLimite de veiculos atingido.\n";
        return;
    }
    Veiculo novo;
    std::cout << "Placa: ";
    std::cin.getline(novo.placa, 10);
    if (buscarVeiculoPorPlaca(novo.placa, veiculos, numVeiculos) != -1) {
        std::cout << "Placa ja cadastrada.\n";
        return;
    }
    std::cout << "Modelo: ";
    std::cin.getline(novo.modelo, MAX_NOME);
    std::cout << "Local atual: ";
    std::cin.getline(novo.localAtual, MAX_NOME);
    novo.status = DISPONIVEL;
    veiculos[numVeiculos++] = novo;
    std::cout << "Veiculo cadastrado.\n";
}

void atualizarVeiculo(Veiculo veiculos[], int numVeiculos, const Local locais[], int
numLocais) {
    char placa[10];
    std::cout << "Placa do veiculo: ";
```

```

std::cin.getline(placa, 10);
int idx = buscarVeiculoPorPlaca(placa, veiculos, numVeiculos);
if (idx == -1) {
    std::cout << "Veiculo nao encontrado.\n";
    return;
}
std::cout << "Novo modelo: ";
std::cin.getline(veiculos[idx].modelo, MAX_NOME);
std::cout << "Veiculo atualizado.\n";
}

void excluirVeiculo(Veiculo veiculos[], int &numVeiculos) {
    char placa[10];
    std::cout << "Placa do veiculo: ";
    std::cin.getline(placa, 10);
    int idx = buscarVeiculoPorPlaca(placa, veiculos, numVeiculos);
    if (idx == -1) {
        std::cout << "Nao encontrado.\n";
        return;
    }
    for (int i = idx; i < numVeiculos - 1; ++i)
        veiculos[i] = veiculos[i + 1];
    numVeiculos--;
    std::cout << "Excluido com sucesso.\n";
}

```

Veiculos.h:

```

#ifndef VEICULOS_H
#define VEICULOS_H

#include "slem.h"

// Declarações das funções que estão em veiculos.cpp
void cadastrarVeiculo(Veiculo veiculos[], int &numVeiculos, const Local locais[], int numLocais);
void listarVeiculos(const Veiculo veiculos[], int numVeiculos);
void atualizarVeiculo(Veiculo veiculos[], int numVeiculos, const Local locais[], int numLocais);
void excluirVeiculo(Veiculo veiculos[], int &numVeiculos);
int buscarVeiculoPorPlaca(const char* placa, const Veiculo veiculos[], int numVeiculos);

#endif

```

Minunit.h:

```

#ifndef MINUNIT_H
#define MINUNIT_H

#define mu_assert(message, test) do { if (!(test)) return message; } while (0)
#define mu_run_test(test) do { char *message = test(); tests_run++; if (message) return message; } while (0)
extern int tests_run;

#endif

```

Test_locais.cpp:

```
#include <stdio>
#include <cstring>
#include "minunit.h"
#include "../src/locais.h"

int tests_run = 0;

char* test_buscarLocalPorNome() {
    Local locais[2];
    strcpy(locais[0].nome, "Centro");
    locais[0].coordX = 0;
    locais[0].coordY = 0;

    strcpy(locais[1].nome, "Bairro");
    locais[1].coordX = 1;
    locais[1].coordY = 1;

    mu_assert("Erro: Centro deve estar na posicao 0", buscarLocalPorNome("Centro", locais,
2) == 0);
    mu_assert("Erro: Bairro deve estar na posicao 1", buscarLocalPorNome("Bairro", locais,
2) == 1);
    mu_assert("Erro: Inexistente deve retornar -1", buscarLocalPorNome("Inexistente",
locais, 2) == -1);
    return 0;
}

char* all_tests() {
    mu_run_test(test_buscarLocalPorNome);
    return 0;
}

int main() {
    char* result = all_tests();
    if (result != 0) {
        printf("FALHOU: %s\n", result);
    } else {
        printf("TODOS OS TESTES PASSARAM\n");
    }
    printf("Testes executados: %d\n", tests_run);
    return result != 0;
}
```

Test_pedidos.cpp:

```
#include <stdio>
#include "minunit.h"
#include "../src/pedidos.h"

int tests_run = 0;

char* test_buscarPedidoPorId() {
    Pedido pedidos[2] = {
        {1, "A", "B", 10.0f},
        {2, "C", "D", 20.0f}
    }
}
```



```

    };
    mu_assert("Erro: Pedido 1 deve estar na posicao 0", buscarPedidoPorId(1, pedidos, 2) ==
0);
    mu_assert("Erro: Pedido 2 deve estar na posicao 1", buscarPedidoPorId(2, pedidos, 2) ==
1);
    mu_assert("Erro: Pedido inexistente deve retornar -1", buscarPedidoPorId(3, pedidos, 2)
== -1);
    return 0;
}

char* all_tests() {
    mu_run_test(test_buscarPedidoPorId);
    return 0;
}

int main() {
    char* result = all_tests();
    if (result != 0) {
        printf("FALHOU: %s\n", result);
    } else {
        printf("TODOS OS TESTES PASSARAM\n");
    }
    printf("Testes executados: %d\n", tests_run);
    return result != 0;
}

```

Test_rota.cpp:

```

#include <stdio>
#include "minunit.h"
#include "../src/rota.h"

int tests_run = 0;

char* test_calcularDistancia() {
    Local l1 = {"A", 0, 0};
    Local l2 = {"B", 3, 4};
    double dist = calcularDistancia(l1, l2);
    mu_assert("Erro: distancia entre (0,0) e (3,4) deve ser 5.0", (int)(dist * 10) == 50);
    return 0;
}

char* all_tests() {
    mu_run_test(test_calcularDistancia);
    return 0;
}

int main() {
    char* result = all_tests();
    if (result != 0) {
        printf("FALHOU: %s\n", result);
    } else {
        printf("TODOS OS TESTES PASSARAM\n");
    }
    printf("Testes executados: %d\n", tests_run);
}

```

```
    return result != 0;
}
```

Test_veiculos.cpp:

```
#include <stdio>
#include <cstring>
#include "minunit.h"
#include "../src/veiculos.h"

int tests_run = 0;

char* test_buscarVeiculoPorPlaca() {
    Veiculo veiculos[2] = {
        {"ABC1234", "Modelo A", DISPONIVEL, "Centro"},
        {"XYZ9876", "Modelo B", OCUPADO, "Bairro"}
    };
    mu_assert("Erro: ABC1234 deve estar na posicao 0", buscarVeiculoPorPlaca("ABC1234",
veiculos, 2) == 0);
    mu_assert("Erro: XYZ9876 deve estar na posicao 1", buscarVeiculoPorPlaca("XYZ9876",
veiculos, 2) == 1);
    mu_assert("Erro: Placa inexistente deve retornar -1", buscarVeiculoPorPlaca("ZZZ9999",
veiculos, 2) == -1);
    return 0;
}

char* all_tests() {
    mu_run_test(test_buscarVeiculoPorPlaca);
    return 0;
}

int main() {
    char* result = all_tests();
    if (result != 0) {
        printf("FALHOU: %s\n", result);
    } else {
        printf("TODOS OS TESTES PASSARAM\n");
    }
    printf("Testes executados: %d\n", tests_run);
    return result != 0;
}
```