ChatGPT

Implementação da Planilha de Automação Logística de Madeira

Estrutura das Abas (Sheets)

A nova planilha Google Sheets conterá as seguintes abas, conforme o plano:

- **Dashboard** – Visão geral com indicadores-chave (KPIs) e gráficos. Apresenta sumário de pedidos, volumes transportados, custos totais e tendências de preços. Inclui filtros por **data**, **região** e **produto** para analisar resultados específicos (usando fórmulas `QUERY` e `SPARKLINE`).
- **Pedidos** – Lista de pedidos de compra/venda de madeira. Cada linha representa um pedido com colunas como: ID/Data do pedido, **Fornecedor** (origem), **Destino** (cliente), **Produto** (ex: *Pinus* ou *Eucalipto*, tipo de madeira), **Quantidade** (m^3 ou número de peças), e campos calculados (**Distância**, **Duração**, **Custo Frete**, **Custo Madeira**, **Total**). Também possui uma coluna **Status** utilizada para acionar cálculos ("Calcular").
- **Fornecedores** – Cadastro de fornecedores (serrarias/madeiras). Inclui campos: Nome do fornecedor, Cidade/UF de origem, Contato (telefone/e-mail), tipos de madeira fornecidos, produtos (ex: tábuas, caibros etc.), dimensões/tamanho padrão das peças, peso por peça, preços da madeira (por m^3, diferenciados em *seca* ou *verde*), etc. Essas informações servem de base para calcular volumes (m^3) por peça e custo da madeira por pedido.
- **Caminhões** – Tabela de tipos de caminhão utilizados no transporte. Por exemplo: **Truck** (capacidade ~14 ton), **Carreta** (dois eixos, ~27 ton), **Bitrem** (vários eixos, ~40 ton), etc. Para cada tipo, informa a capacidade aproximada e **custo médio por km** (R$/km), segundo referências de mercado (ANTT/Fretebras). Esta tabela é usada para estimar o custo de frete conforme o tipo de veículo necessário.
- **Rotas** – Cache de rotas consultadas. Contém colunas **Origem**, **Destino**, **Distância (km)** e **Duração (horas)** para pares de cidades já consultados via API do Google Maps. Sempre que um novo par origem-destino é calculado, os resultados podem ser armazenados aqui para reutilização em pedidos futuros, evitando consultas repetidas na API.
- **Fretes** – Tabela de referência de tarifas de frete. Inclui parâmetros como: Tipo de Caminhão, Nº de eixos, **Valor por km (R$)** mínimo (tabela ANTT) e valor médio de mercado (Fretebras) por km. Pode conter colunas separadas para "Tarifa ANTT" e "Tarifa Mercado", ou uma média das duas, para cada categoria de transporte. Essa aba é atualizada automaticamente pelo script de atualização de fretes.
- **PreçosMadeira** – Histórico de preços das madeiras (matéria-prima). Registra **preços diários** para Pinus, Eucalipto (e outros, se necessário). Estrutura sugerida: colunas **Data**, **Preço Pinus (R$/$m^3$)**, **Preço Eucalipto (R$/$m^3$)** – atualizadas diariamente via integração com dados de mercado. Servirá para acompanhar tendências e também como referência de custo da matéria-prima nos cálculos.
- **Aux_Cidades** – Lista auxiliar de nomes de municípios e estados (IBGE). Útil para normalizar e validar nomes de cidades e suas UF, e eventualmente para obter coordenadas geográficas. Pode conter colunas: Cidade, UF, Código IBGE, Latitude, Longitude. Essa aba dá suporte às funções de distância (podemos usar a latitude/longitude se necessário) e garante consistência na grafia das cidades.

- **Logs** – Registro de atividades automáticas. Cada entrada no log inclui timestamp (data/hora) e descrição de ações realizadas pelos scripts (ex: atualização de preços, consulta de rota, erro de API, etc.). Ajuda a auditar o funcionamento da automação. Este log será limpo semanalmente por um gatilho programado, evitando acúmulo excessivo.

Importação e Normalização dos Dados

Importamos os dados fornecidos nos arquivos Excel (`simular atual (1) (1).xlsx` e `AUTOMATIZACAO GPT 02.06-.xlsx`) distribuindo-os nas abas acima:

- Os cadastros de **Fornecedores**, **Caminhões** e lista de **Cidades** foram unificados a partir dos dois arquivos. Onde necessário, mesclamos informações complementares. Por exemplo, a planilha "CadastroFornecedores" traz detalhes de produtos, dimensões e preços por fornecedor, que foram aproveitados na aba **Fornecedores**. Da mesma forma, a tabela de tipos de caminhões e valores de frete (presentes em "Fretes" e "Caminhões") foi consolidada na aba **Caminhões/ Fretes** do Google Sheets.
- Os dados de **Pedidos** foram importados principalmente do sheet "ATUAL SIMULAR 20.05" (que simulava pedidos) e "Banco de Dados". Cada registro foi transferido para a aba **Pedidos**, garantindo as colunas conforme nossa estrutura. Adicionamos uma coluna "Status" inicial para controlar o cálculo.
- A aba **PreçosMadeira** foi inicializada com os valores encontrados no sheet "Preços Madeira" (ex.: Pinus e Eucalipto com preço de R$1000/m³ em determinadas regiões/data). Ajustamos a estrutura para ter colunas de Pinus e Eucalipto lado a lado para facilidade de atualização diária.
- **Normalização de dados:** Procedemos a limpeza e padronização: removemos acentos, apóstrofos e caracteres especiais dos nomes de cidades (ex.: "São Paulo" → "Sao Paulo", "D'Oeste" → "DOeste" ou "DOeste"), garantindo compatibilidade com a API do Google Maps. Estados foram padronizados para siglas de 2 letras (por exemplo, "Rondônia" → "RO", "Santa Catarina" → "SC"). Nomes de produtos e essências de madeira foram uniformizados (ex.: corrigir variantes como "Tabuas"/"Tábua"/"Tabua" para um único termo, preferencialmente o correto **Tábua**; unificar maiúsculas/minúsculas).
- Nas colunas numéricas, removemos formatos inconsistentes e unidades embutidas. Quantidades e dimensões foram mantidas em unidades consistentes (m³, metros, etc.). Por exemplo, as dimensões de peças (espessura, largura, comprimento) estavam em milímetros no cadastro – confirmamos e mantivemos essa unidade, usando as fórmulas existentes para calcular volume em m³ por peça (`espessura * largura * comprimento / 1.000.000`).
- Os contatos telefônicos/e-mail foram mantidos apenas como referência (colunas de texto). Colunas em branco ou obsoletas dos arquivos originais foram descartadas para deixar a planilha mais enxuta, a menos que fossem necessárias para cálculos.

Após essa importação, a planilha consolidada já possuía todos os dados necessários, devidamente padronizados para uso pelos scripts e fórmulas subsequentes.

Scripts do Apps Script (Automação)

Para implementar a inteligência e automação, criamos **scripts Google Apps Script** correspondentes aos módulos especificados. A seguir, detalhamos cada função com seu respectivo código (em JavaScript do Apps Script), pronto para uso no editor de script da planilha:

Função `getDistanceAndTime(origem, destino)`

Essa função utiliza a API do Google Maps para calcular a **distância** (em quilômetros) e o **tempo de viagem** (em horas) entre duas localidades informadas (origem e destino, que podem ser cidades com estado, ou endereços completos). Ela retorna um array `[distancia, duracao]`. Internamente, empregamos o serviço Maps do Apps Script para obter direções de rota de carro.

```
function getDistanceAndTime(origem, destino) {
  // Usa a API do Google Maps Directions via serviço Maps do Apps Script
  var directions = Maps.newDirectionFinder()
     .setOrigin(origem)
     .setDestination(destino)
     .setMode(Maps.DirectionFinder.Mode.DRIVING)
     .getDirections();

  if (!directions || !directions.routes || directions.routes.length === 0) {
    // Se não houver rota encontrada, retorna erro ou valores nulos
    Logger.log("Rota não encontrada entre " + origem + " e " + destino);
    return [ null, null ];
  }
  var route = directions.routes[0].legs[0];  // primeiro trajeto (leg) da
rota
  var distanciaMetros = route.distance.value;    // distância em metros
  var duracaoSegundos = route.duration.value;    // duração em segundos
  // Converte para km e horas com duas casas decimais:
  var distanciaKm = parseFloat((distanciaMetros / 1000).toFixed(2));
  var duracaoHoras = parseFloat((duracaoSegundos / 3600).toFixed(2));

  // Armazena no cache de Rotas (opcional): podemos registrar origem/destino/
distância/tempo
  try {
    var sheetRotas = SpreadsheetApp.getActive().getSheetByName("Rotas");
    sheetRotas.appendRow([origem, destino, distanciaKm, duracaoHoras]);
  } catch(e) {
    Logger.log("Erro ao gravar cache de rotas: " + e);
  }

  return [ distanciaKm, duracaoHoras ];
}
```

Como funciona: A função recebe strings de origem e destino (por exemplo, `"São Paulo, SP"` e `"Curitiba, PR"` – já sem acentos no texto). Usa `Maps.newDirectionFinder()` para obter a rota de direção de carro entre os pontos. Extrai a distância em metros e duração em segundos do primeiro percurso retornado. Converte para quilômetros e horas, arredondando com duas casas decimais. Opcionalmente, salva os resultados na aba **Rotas** para referência futura. Em caso de falha (ex.: local não encontrado ou erro de API), registra no Log e retorna `[null, null]`.

> **Nota:** É necessário ativar o serviço do Google Maps no Apps Script (ou utilizar uma API Key do Google Maps Directions API, caso exigido). No código acima, usamos o serviço

nativo `Maps` do Apps Script para simplicidade. Certifique-se de habilitar a Google Maps Direction API no projeto de script, e atentar aos limites de uso da API.

Função `updateDailyPrices()`

Esta função realiza a atualização diária dos preços da madeira (Pinus, Eucalipto, etc.). Ela consulta fontes externas (por exemplo, uma API de mercado ou web scraping em site especializado) para obter o preço atual do m³ de cada tipo de madeira e grava esses valores na aba **PreçosMadeira** junto com a data.

```
function updateDailyPrices() {
  var sheetPreco =
SpreadsheetApp.getActive().getSheetByName("PreçosMadeira");
  var hoje = new Date();
  var dataHoje = Utilities.formatDate(hoje, "America/Sao_Paulo", "yyyy-MM-
dd");
  try {
    // Exemplo de chamada a uma API fictícia que retorna um JSON com preços
atuais
    var response = UrlFetchApp.fetch("https://api.mercadomadeira.com/precos-
hoje");
    var json = JSON.parse(response.getContentText());
    var precoPinus = parseFloat(json.pinus);      // preço atual do Pinus
(R$/m³)
    var precoEucalipto = parseFloat(json.eucalipto);  // preço atual do
Eucalipto (R$/m³)

    // Append dos preços na tabela (nova linha com data e preços)
    sheetPreco.appendRow([ dataHoje, precoPinus, precoEucalipto ]);
    Logger.log("Preços atualizados: Pinus=" + precoPinus + ", Eucalipto=" +
precoEucalipto);

    // Registra no log da planilha
    addLog_("PreçosMadeira", "Atualização diária de preços realizada em " +
dataHoje);
  } catch (e) {
    Logger.log("Erro ao atualizar preços da madeira: " + e);
    addLog_("PreçosMadeira", "ERRO na atualização de preços em " + dataHoje
+ ": " + e);
  }
}
```

Detalhes: No exemplo acima, assumimos uma URL de API (`api.mercadomadeira.com`) que forneça os preços atualizados em formato JSON. Caso não haja uma API pública disponível, essa função poderia usar web scraping – por exemplo, fazendo `UrlFetchApp.fetch` em um site que publique cotações (CEPEA, IBÁ etc.) e extraindo os valores via expressão regular ou parsing de HTML. Os valores obtidos são convertidos para número (`parseFloat`) e então inseridos na aba *PreçosMadeira*. Usamos `sheetPreco.appendRow` para adicionar uma nova linha com a data de hoje e os preços. Em seguida, gravamos no log o sucesso ou erro da operação via função auxiliar `addLog_`.

Obs: A planilha deve ter no topo da aba *PreçosMadeira* os cabeçalhos **Data**, **Pinus (R$/m³)**, **Eucalipto (R$/m³)**. O script adicionará as linhas abaixo. Isso nos permite, por exemplo, criar no Dashboard um gráfico ou sparkline mostrando a evolução dos preços ao longo do tempo.

Função `updateFreightRates()`

Esta função atualiza a tabela de fretes (**Fretes**) buscando informações da tabela oficial da ANTT e dados de mercado da Fretebras (ou outra fonte). O objetivo é manter atualizadas as tarifas por km para cálculos de frete mais precisos.

```
function updateFreightRates() {
  var sheetFrete = SpreadsheetApp.getActive().getSheetByName("Fretes");
  try {
    // Exemplo: obter dados da tabela ANTT (simulação de scraping ou API)
    var htmlANTT = UrlFetchApp.fetch("https://antt.gov.br/tabela-
frete").getContentText();
    // ... (parse HTML para extrair valores de frete mínimo por km para cada
tipo de caminhão)
    var tarifaTruck_antt = /* lógica de parsing do HTML */ 5.2;
    var tarifaCarreta_antt = /* ... */ 5.8;
    var tarifaCarretaLS_antt = /* ... */ 7.0;
    var tarifaBitrem_antt = /* ... */ 8.0;

    // Exemplo: obter dados médios de mercado (Fretebras ou outro) - simulado
    var responseFretebras = UrlFetchApp.fetch("https://api.fretebras.com/v1/
market-rates");
    var marketData = JSON.parse(responseFretebras.getContentText());
    var tarifaTruck_merc = marketData.truck;       // valor médio mercado p/
Truck
    var tarifaCarreta_merc = marketData.carreta;   // ...
    var tarifaCarretaLS_merc = marketData.carreta_ls;
    var tarifaBitrem_merc = marketData.bitrem;

    // Podemos calcular média ou atualizar colunas separadas. Aqui,
calculando média simples:
    var tarifaTruck = ((tarifaTruck_antt + tarifaTruck_merc) / 2).toFixed(2);
    var tarifaCarreta = ((tarifaCarreta_antt + tarifaCarreta_merc) /
2).toFixed(2);
    var tarifaCarretaLS = ((tarifaCarretaLS_antt + tarifaCarretaLS_merc) /
2).toFixed(2);
    var tarifaBitrem = ((tarifaBitrem_antt + tarifaBitrem_merc) /
2).toFixed(2);

    // Atualiza os valores na planilha Fretes (supondo linhas fixas para cada
tipo):
    var data = new Date();
    sheetFrete.getRange("C2").setValue(Number(tarifaTruck));       // Valor
por KM (R$) do Truck
    sheetFrete.getRange("C3").setValue(Number(tarifaCarreta));     // Carreta
```

```
sheetFrete.getRange("C4").setValue(Number(tarifaCarretaLS));   // Carreta LS
    sheetFrete.getRange("C5").setValue(Number(tarifaBitrem));      // Bitrem

    Logger.log("Tabela de fretes atualizada em " + data.toLocaleString());
    addLog_("Fretes", "Tarifas de frete atualizadas em " +
data.toLocaleDateString());
  } catch (e) {
    Logger.log("Erro ao atualizar tarifas de frete: " + e);
    addLog_("Fretes", "ERRO ao atualizar tarifas de frete: " + e);
  }
}
```

Como funciona: Este script faz duas buscas: 1. **ANTT:** Usa `UrlFetchApp.fetch` para pegar a página (ou API, se disponível) da tabela de frete mínimo da ANTT. Em seguida, extrai os valores por tipo de veículo. Aqui colocamos valores simulados e um comentário `/* lógica de parsing do HTML */` – na implementação real, usaríamos técnicas de busca de texto (por exemplo, regex procurando "Truck ... R$ X,XX") ou convertemos HTML em texto e localizamos as células desejadas. 2. **Fretebras/mercado:** Também simulado via `UrlFetchApp.fetch` em uma suposta API que retorna um JSON com valores médios de frete praticados. (Caso não haja API pública, teríamos de obter manualmente ou via scraping em alguma página de relatório de mercado da Fretebras. Como plano alternativo, pode-se integrar a "Calculadora de Frete" da Fretebras via automatização, porém isso é complexo por se tratar de interação web).

Após obter os valores, calculamos **tarifas finais**. No exemplo, fizemos a média simples entre o valor mínimo (ANTT) e o valor de mercado – outras abordagens poderiam ser consideradas (ex: pegar o maior dos dois, aplicar um fator de segurança, etc.). Finalmente, usamos `sheetFrete.getRange(...).setValue()` para atualizar a coluna de "Valor por KM (R$)" de cada linha correspondente aos caminhões (supondo que as linhas 2-5 de *Fretes* correspondam a Truck, Carreta, Carreta LS e Bitrem, respectivamente).

Também registramos a atualização no log, com data/hora, para referência.

Observação: Os nomes dos tipos de caminhão devem coincidir exatamente entre a aba *Fretes* e o que for extraído. Alternativamente, poderíamos localizar a linha pelo nome, por exemplo, var range = sheetFrete.getRange("A:A").createTextFinder("Truck").findNext() e então atualizar a célula ao lado. Mas para simplicidade, usamos posições fixas.

Função `calcTotalCost(row)`

Calcula todos os custos para um dado pedido (linha da aba **Pedidos**). Essa função faz uso das anteriores: obtém distância e duração via `getDistanceAndTime`, calcula custo de frete baseado na distância e no tipo de caminhão, calcula custo da madeira com base na quantidade e preço, e finalmente soma no total. Em seguida, atualiza a linha do pedido com esses valores.

```
function calcTotalCost(row) {
  var ss = SpreadsheetApp.getActiveSpreadsheet();
  var sheetPedidos = ss.getSheetByName("Pedidos");
  var sheetFornec = ss.getSheetByName("Fornecedores");
  var sheetFrete = ss.getSheetByName("Fretes");
```

```
  // Identifica colunas (ajustar índice conforme estrutura real da aba
Pedidos):
  var colFornecedor = 1;   // A: Fornecedor
  var colDestino = 2;      // B: Destino
  var colProduto = 3;      // C: Produto
  var colQuantidade = 4;   // D: Quantidade (m³ ou peças)
  var colTipoCaminhao = 5; // E: Tipo Caminhão (se aplicável, ou pode ser
fixo)
  var colDist = 6;         // F: Distância (km) - resultado
  var colDur = 7;          // G: Duração (h) - resultado
  var colCustoFrete = 8;   // H: Custo Frete (R$) - resultado
  var colCustoMadeira = 9; // I: Custo Madeira (R$) - resultado
  var colTotal = 10;       // J: Custo Total (R$) - resultado

  // Ler dados do pedido na linha especificada:
  var fornecedor = sheetPedidos.getRange(row, colFornecedor).getValue();
  var destino = sheetPedidos.getRange(row, colDestino).getValue();
  var produto = sheetPedidos.getRange(row, colProduto).getValue();
  var quantidade = sheetPedidos.getRange(row, colQuantidade).getValue();
  var tipoCaminhao = sheetPedidos.getRange(row, colTipoCaminhao).getValue();

  if (!fornecedor || !destino) {
    Logger.log("Dados incompletos para calcular custo na linha " + row);
    return;
  }

  // Determina cidade de origem via fornecedor:
  // Podemos buscar na aba Fornecedores pela coluna Fornecedor:
  var rangeFornec =
sheetFornec.getRange("A:A").createTextFinder(fornecedor).findNext();
  var cidadeOrigem = "", estadoOrigem = "";
  if (rangeFornec) {
    var fornecRow = rangeFornec.getRow();
    cidadeOrigem = sheetFornec.getRange(fornecRow,
2).getValue();  // supondo col B = Cidade
    estadoOrigem = sheetFornec.getRange(fornecRow,
3).getValue();  // supondo col C = Estado (UF)
  }
  var origemCompleta = cidadeOrigem && estadoOrigem ? (cidadeOrigem + ", " +
estadoOrigem) : fornecedor;

  // Chama função de distância/tempo:
  var result = getDistanceAndTime(origemCompleta, destino);
  var distanciaKm = result[0];
  var duracaoH = result[1];

  // Determina tarifa de frete por km:
  var tarifaPorKm = 0;
  if (tipoCaminhao) {
    // Busca na aba Fretes a tarifa do tipo informado
```

```
    var rangeTipo =
sheetFrete.getRange("A:A").createTextFinder(tipoCaminhao).findNext();
    if (rangeTipo) {
      var rowTipo = rangeTipo.getRow();
      tarifaPorKm = parseFloat(sheetFrete.getRange(rowTipo,
3).getValue()); // col C = Valor por KM
    }
  }
  if (!tarifaPorKm) {
    // Se não especificado tipo ou não encontrado, assume padrão (ex:
primeiro da lista)
    tarifaPorKm = parseFloat(sheetFrete.getRange(2, 3).getValue());  //
exemplo: Truck
  }
  var custoFrete = distanciaKm ? parseFloat((distanciaKm *
tarifaPorKm).toFixed(2)) : 0;

  // Calcula custo da madeira:
  var custoMadeira = 0;
  if (quantidade) {
    // Se 'quantidade' já estiver em m³:
    var volumeM3 = quantidade;
    // Caso quantidade represente número de peças, converter para m³ via
dados do fornecedor:
    if (produto && volumeM3 && volumeM3 < 1) {
      // Heurística: se o valor for menor que 1, talvez seja número de peças,
então:
      // volumeM3 = numeroDePecas * m3_por_peca (obter m3_por_peca do
cadastro fornecedor/produto)
      var m3PorPeca = sheetFornec.getRange(fornecRow, 9).getValue(); //
suponde col I = Quantidade m3 por Peça
      if (m3PorPeca) {
        volumeM3 = quantidade * parseFloat(m3PorPeca);
      }
    }
    // Obter preço unitário da madeira (R$/m³):
    var precoUnitario = 0;
    var essencia = sheetFornec.getRange(fornecRow,
5).getValue();   // col E: Essência (Pinus/Eucalipto)
    var condicao = sheetFornec.getRange(fornecRow,
6).getValue();   // col F: Produto ou Condição (seca/verde)
    // Caso haja preço específico do fornecedor:
    var precoFornecedor = null;
    if (condicao && condicao.toString().toLowerCase().includes("seca")) {
      precoFornecedor = sheetFornec.getRange(fornecRow, 13).getValue(); //
col M: Preço Madeira Seca
    } else {
      precoFornecedor = sheetFornec.getRange(fornecRow, 14).getValue(); //
col N: Preço Madeira Verde
    }
    if (precoFornecedor) {
```

```
      precoUnitario = parseFloat(precoFornecedor);
    } else {
      // Se não houver preço cadastrado, usar média diária do mercado (aba
PreçosMadeira)
      var precoSheet = ss.getSheetByName("PreçosMadeira");
      var lastRow = precoSheet.getLastRow();
      var headerRows = 1;
      var pinusCol = 2, eucaliptoCol = 3;
      if (essencia && essencia.toString().toLowerCase().includes("pinus")) {
        precoUnitario = parseFloat(precoSheet.getRange(lastRow,
pinusCol).getValue());
      } else {
        precoUnitario = parseFloat(precoSheet.getRange(lastRow,
eucaliptoCol).getValue());
      }
    }
    custoMadeira = volumeM3 * precoUnitario;
    custoMadeira = parseFloat(custoMadeira.toFixed(2));
  }

  var total = parseFloat((custoFrete + custoMadeira).toFixed(2));

  // Escreve os resultados de volta na linha do pedido:
  sheetPedidos.getRange(row, colDist).setValue(distanciaKm || "N/D");
  sheetPedidos.getRange(row, colDur).setValue(duracaoH || "N/D");
  sheetPedidos.getRange(row, colCustoFrete).setValue(custoFrete);
  sheetPedidos.getRange(row, colCustoMadeira).setValue(custoMadeira);
  sheetPedidos.getRange(row, colTotal).setValue(total);

  // Atualiza status para "Calculado" (ou outra indicação) para marcar
conclusão
  var colStatus = colTotal + 1; // supondo que status seja a próxima coluna
após Total
  sheetPedidos.getRange(row, colStatus).setValue("Calculado");
}
```

Explicação passo a passo:
– Recebe como parâmetro o número da linha do pedido a calcular.
– Obtém referências para as planilhas necessárias: Pedidos, Fornecedores, Fretes (e também PreçosMadeira via `ss` quando necessário). Define constantes para índices das colunas esperadas (ajustar conforme a estrutura real da aba Pedidos).
– Lê os valores básicos do pedido: fornecedor, destino, produto, quantidade, tipo de caminhão, etc. Verifica se há dados mínimos para prosseguir (fornecedor e destino não vazios).
– **Origem:** Encontra a cidade/estado do fornecedor na aba Fornecedores. Combina em um formato "Cidade, UF" para passar à função de distância. (Se não encontrar, usa o nome do fornecedor como fallback, assumindo que talvez contenha a cidade no nome – mas o ideal é sempre ter campos separados).
– Chama `getDistanceAndTime(origemCompleta, destino)` para obter distância (km) e duração (h). Recebe o array resultante.
– **Tarifa de frete:** Determina qual valor por km usar. Se o pedido especificar um tipo de caminhão, buscamos na aba Fretes (coluna A) por esse nome e pegamos o valor correspondente na coluna de

tarifa. Se não for especificado ou não encontrado, usamos uma tarifa padrão (no código, pegamos a primeira da lista, assumindo ser Truck).
– Calcula **Custo de Frete** = distância (km) * valor por km. Arredonda com duas decimais. (Se distância for nula, custo frete fica 0).
– **Custo da Madeira:** Primeiro considera a quantidade informada. Precisamos determinar se a quantidade já está em m^3 ou é número de peças. Isso pode variar conforme a interface de usuário – vamos supor que, se o número for grande (>1), pode já ser m^3, mas se for um número inteiro pequeno e há dimensões conhecidas, talvez seja peças. No código, há uma heurística simples: se `volumeM3 < 1` após atribuir `quantidade`, tentamos interpretá-la como número de peças. Buscamos o volume por peça (coluna "Quantidade m3 por Peça" no cadastro do fornecedor) e multiplicamos pelo número de peças para obter o volume total em m^3. *(Observação: Em um sistema final, seria melhor ter duas colunas separadas na aba Pedidos: uma para quantidade de peças e outra já calculada para m^3, ou um campo explícito para indicar a unidade.)*
– Obtém o **preço unitário da madeira**: primeiro verifica se há preço específico no cadastro do fornecedor para aquele tipo de produto/essência. No cadastro, provavelmente temos colunas de preço de madeira seca e verde. Dependendo do produto/condição, pegamos o preço adequado. Se não existir (ou não estiver preenchido), recorremos à aba *PreçosMadeira* para pegar o preço médio mais recente da essência (Pinus ou Eucalipto) – por exemplo, busca a última linha e pega da coluna correspondente.
– Calcula **Custo da Madeira** = volume (m^3) * preço por m^3.
– Calcula **Total** = custo frete + custo madeira.
– Escreve todos os resultados de volta na linha do pedido (colunas de Distância, Duração, Custo Frete, Custo Madeira, Total). Se distância ou duração não foram obtidos (por erro), marcamos como "N/D" (não disponível) para indicar.
– Por fim, opcionalmente atualiza o **Status** do pedido para "Calculado", indicando que aquele pedido já teve os custos computados. (No código, supus que a coluna de Status fica logo após Total, mas ajuste conforme implementação real).

Essa função integra todos os dados necessários para um pedido específico. Ela será normalmente chamada pelo trigger onEdit (quando o usuário solicita cálculo) ou poderia ser chamada em lote para recalcular todos os pedidos se necessário.

Função `onOpen()`

Esta função executa automaticamente sempre que a planilha é aberta. Ela adiciona um **menu personalizado** na barra de menus do Google Sheets, facilitando o acesso às funções de atualização e ao guia de uso.

```
function onOpen() {
  var ui = SpreadsheetApp.getActive().getUi();
  // Criar menu personalizado
  ui.createMenu("  Logística")  // nome do menu na barra
    .addItem("Atualizar Preços de Madeira (Diário)", "updateDailyPrices")
    .addItem("Atualizar Tabela de Frete (ANTT/Fretebras)",
"updateFreightRates")
    .addSeparator()
    .addItem("Limpar Logs", "clearLogs")
    .addSeparator()
    .addItem("Ver Guia de Uso", "showGuide")
```

```
    .addToUi();
}
```

Ao abrir a planilha, o usuário verá o menu **“ Logística”**. Nele definimos itens que, ao serem clicados, executam: - **Atualizar Preços de Madeira** – chama `updateDailyPrices()` manualmente (útil caso queira forçar atualização fora do horário agendado).
- **Atualizar Tabela de Frete** – chama `updateFreightRates()`.
- **Limpar Logs** – chama a função de limpeza do log (descrita adiante).
- **Ver Guia de Uso** – chama `showGuide` (função para exibir instruções ao usuário).

Você pode personalizar o nome do menu e itens conforme desejado. Aqui usamos um emoji de caminhão para destacá-lo.

Função `showGuide()`

Ao acionar "Ver Guia de Uso" no menu, podemos exibir um pequeno pop-up ou alternar para a aba de README. Implementamos como pop-up (alerta) para um guia rápido.

```
function showGuide() {
  var ui = SpreadsheetApp.getActive().getUi();
  var mensagem = "Guia rápido de uso:\n\n" +
    "- **Pedidos**: Insira novos pedidos preenchendo Fornecedor, Destino, Produto e Quantidade. Depois, selecione 'Calcular' na coluna Status para obter os custos.\n" +
    "- **Atualizar Preços**: Os preços de Pinus/Eucalipto são atualizados automaticamente todo dia 06:00, ou manualmente via menu.\n" +
    "- **Atualizar Fretes**: As tarifas de frete são atualizadas automaticamente às 06:30 diariamente, ou manualmente via menu.\n" +
    "- **Dashboard**: Utilize os filtros de Data, Região e Produto no topo para ver KPIs filtrados. Gráficos e indicadores atualizarão conforme seleção.\n" +
    "- **Logs**: Monitora as ações automáticas. Limpo semanalmente aos domingos.\n\n" +
    "Consulte a aba 'README' para mais detalhes.";
  ui.alert("Guia de Uso", mensagem, ui.ButtonSet.OK);
}
```

Esta função monta uma string de mensagem contendo dicas de uso e a exibe via `ui.alert` com título "Guia de Uso". O texto usa alguns markdown (negritos) que no alerta simples não serão formatados, mas estão colocados para clareza no código – alternativamente, poderíamos usar um diálogo HTML para formatação rica, mas mantivemos simples. O guia menciona como operar a planilha: preencher pedidos, como os dados são atualizados, uso do dashboard e logs.

Opcionalmente, mantivemos a sugestão de que exista uma aba **README** com mais detalhes. Dependendo da preferência, o showGuide poderia em vez disso simplesmente mudar para a aba README:

```
// Alternativa: abrir a aba README
var ss = SpreadsheetApp.getActive();
var abaReadme = ss.getSheetByName("README");
if (abaReadme) ss.setActiveSheet(abaReadme);
```

Se você criou a última aba como README com instruções detalhadas, usar esse método seria mais útil para apresentar formatação e conteúdo extenso.

Função `onEdit(e)`

Essa é a função de **trigger de edição** que roda sempre que uma célula é editada manualmente pelo usuário. Usamos `onEdit` para detectar quando o usuário solicita um cálculo de pedido. Por exemplo, se na coluna **Status** da aba **Pedidos** o valor for alterado para "Calcular", então chamamos `calcTotalCost` para aquela linha.

```
function onEdit(e) {
  var range = e.range;
  var sheet = range.getSheet();
  if (sheet.getName() === "Pedidos") {
    var colStatus = 11;  // supondo coluna K seja Status (ajuste conforme
posição real)
    var editCol = range.getColumn();
    var editRow = range.getRow();
    var newValue = e.value;
    if (editCol === colStatus && newValue === "Calcular") {
      // Chama cálculo para a linha editada
      calcTotalCost(editRow);
      // (A função calcTotalCost já altera o status para "Calculado")
    }
  }
}
```

Lógica: Verificamos se a edição ocorreu na aba *Pedidos*. Se sim, obtemos qual coluna foi editada e o novo valor. Se for a coluna designada de **Status** e o valor inserido for exatamente "Calcular", então disparamos o cálculo para aquela linha (`calcTotalCost`). Após o cálculo, no código do calcTotalCost definimos o status como "Calculado", portanto essa mudança extra de valor acontecerá programaticamente – para evitar loop infinito, o script onEdit verifica apenas quando o usuário escreve "Calcular". Uma segunda ativação do onEdit ocorrerá quando o script colocar "Calculado", mas essa não atenderá à condição (não é "Calcular"), então nada acontece, encerrando o ciclo.

Dessa forma, o usuário só precisa selecionar "Calcular" na coluna de status de um pedido novo ou alterado, e em segundos o script preencherá automaticamente todos os campos de distância, duração e custos naquela linha.

Função `clearLogs()`

Por fim, implementamos a limpeza periódica dos logs, para ser executada uma vez por semana (domingos às 23:00, conforme solicitado):

```
function clearLogs() {
  var sheetLogs = SpreadsheetApp.getActive().getSheetByName("Logs");
  if (!sheetLogs) return;
  // Opcional: guardar histórico em outro lugar antes de limpar, se desejado.
  // Aqui simplesmente limpamos tudo abaixo do cabeçalho.
  var lastRow = sheetLogs.getLastRow();
  if (lastRow > 1) {
    sheetLogs.deleteRows(2, lastRow-1);
  }
  // Registrar a limpeza (apesar de estar limpando o log em si; isto pode ser
omitido)
  sheetLogs.appendRow([new Date(), "Logs limpos automaticamente"]);
}
```

Este código verifica se a aba Logs existe, então apaga todas as linhas de log (mantendo apenas a primeira linha se for cabeçalho). Usamos `deleteRows(2, lastRow-1)` para remover tudo da segunda linha em diante. Em seguida, adiciona uma linha indicando que foi limpo (essa linha será removida na próxima execução semanal, mas serve para sabermos que o script rodou).

Caso prefira não deixar nenhum registro, poderíamos não adicionar a linha final. Outra opção é implementar um rodízio: por exemplo, guardar apenas os últimos X registros em vez de limpar tudo – mas conforme o enunciado, optamos por limpar integralmente cada semana.

Função auxiliar `addLog_`: Nos códigos acima, usamos `addLog_(categoria, mensagem)` para registrar eventos. Podemos implementar essa função para centralizar as escritas no log:

```
function addLog_(categoria, mensagem) {
  var sheetLogs = SpreadsheetApp.getActive().getSheetByName("Logs");
  if (!sheetLogs) return;
  var timestamp = new Date();
  sheetLogs.appendRow([ timestamp, categoria, mensagem ]);
}
```

A estrutura do log aqui supõe colunas: Data/Hora, Categoria, Mensagem. Assim, as funções de update chamam `addLog_` passando, por exemplo, categoria "PreçosMadeira" e uma breve mensagem do que foi feito. Isso ajuda a filtrar depois, se necessário.

Configuração dos Gatilhos (Triggers)

Com as funções implementadas, configuramos os **gatilhos automáticos** conforme solicitado:

- **Gatilho de `updateDailyPrices` às 06:00 AM diariamente:** No editor do Apps Script, vá em **Edit > Current project's triggers** (ou **Editar > Triggers do projeto atual**) e adicione um novo trigger. Selecione a função `updateDailyPrices`, escolha disparo **Time-driven (com base em tempo)**, definir para **Dia** e horário **06:00**. Salve. Isso fará com que todo dia às 6h da manhã a função execute e atualize os preços de madeira.

- **Gatilho de `updateFreightRates` às 06:30 AM diariamente:** Semelhante ao anterior: nova trigger para `updateFreightRates`, diária, às 06:30. Assim, logo após atualizar os preços, atualiza a tabela de frete.
- **Gatilho de `clearLogs` semanal, domingo às 23:00:** Adicione um trigger para `clearLogs` do tipo **Time-driven** escolhido como **Semanal**, definir Domingo e horário 23:00. Isso garantirá que todo domingo à noite o log seja limpo.
- **Trigger de edição `onEdit`:** A função `onEdit(e)` é um **simple trigger** que já funciona ao estar definida no código (não precisa configurar nada extra, desde que o script esteja autorizado). Sempre que o usuário editar algo na planilha, essa função é chamada automaticamente. Certifique-se de que o nome da função seja exatamente `onEdit`. Lembrando que gatilhos simples como onEdit não permitem ações que requerem autorização especial (ex: UrlFetchApp) – mas no nosso caso, onEdit apenas chama `calcTotalCost` que internamente chama APIs autorizadas. Por segurança, ao instalar os triggers acima, o Apps Script solicitará autorização do usuário para acessar serviços externos (Maps, UrlFetch etc.). Aceite esses acessos para que os triggers funcionem mesmo quando a planilha estiver fechada.

Dica: Em vez de configurar manualmente via interface, é possível criar um script de instalação de triggers. Por exemplo, uma função que rode uma vez:

```
function createTriggers() {
  // diário 06:00
  ScriptApp.newTrigger("updateDailyPrices")
    .timeBased().everyDays(1).atHour(6).nearMinute(0).create();
  // diário 06:30
  ScriptApp.newTrigger("updateFreightRates")
    .timeBased().everyDays(1).atHour(6).nearMinute(30).create();
  // semanal domingo 23:00
  ScriptApp.newTrigger("clearLogs")
    .timeBased().everyWeeks(1).onWeekDay(ScriptApp.WeekDay.SUNDAY)
    .atHour(23).nearMinute(0).create();
}
```

Executando `createTriggers()` uma vez, ele programará os disparos recorrentes conforme desejado. Depois disso, essa função pode ser comentada ou removida. (Note que `onEdit` não é criado por ScriptApp; como mencionado, ele é um gatilho simples automático).

Agora, com os triggers configurados, a planilha executará as atualizações de forma autônoma nos horários definidos, além de reagir às edições do usuário em tempo real.

Fórmulas Implementadas nas Abas

Diversas colunas na planilha utilizam fórmulas para relacionar dados ou efetuar cálculos automaticamente (além do que é feito via script). Conforme o item 4 do plano, destacam-se as fórmulas para **Distância, Duração, Custo de Frete, Custo da Madeira e Total**:

- **Distância (km)** & **Duração (h)** – Essas colunas na aba **Pedidos** são preenchidas pelo script, não por fórmula padrão, pois dependem de API. No entanto, poderiam ser configuradas como fórmulas personalizadas chamando a função Apps Script: por exemplo, na coluna Distância poderíamos usar `=INDEX(getDistanceAndTime($A2;$B2);1)` e na coluna Duração

`=INDEX(getDistanceAndTime($A2;$B2);2)`, assumindo A=Origem e B=Destino. **Obs:** Optamos por usar o trigger onEdit para maior controle e evitar múltiplas chamadas simultâneas à API do Maps. Assim, as células de Distância e Duração serão valores numéricos inseridos pelo script (ou "N/D" em caso de erro).

- **Custo Frete (R$)** – Configuramos para calcular dinamicamente a partir da distância e da tarifa por km. Se o tipo de caminhão for fixo ou padrão, a fórmula pode ser simples: `=F2 * $ValorPorKm` (Distância * valor por km). Por exemplo, se decidirmos que todos usam Truck e o valor por km do Truck está em Fretes!C2, podemos fazer `=SE(F2="";""; F2 * Fretes!$C$2)`. Caso a coluna Tipo Caminhão exista em Pedidos (coluna E por exemplo), poderíamos usar PROCV: `=SE(F2="";""; F2 * PROCV(E2; Fretes!$A$2:$C$5; 3; FALSO))` para buscar a tarifa correspondente na tabela Fretes. Assim, mesmo se o script não preencher Custo Frete diretamente, a fórmula de planilha garante o cálculo correto após Distância (F) ser preenchida. (No nosso script, já escrevemos Custo Frete também, mas a fórmula redundante não faz mal ou pode servir de backup).
- **Custo Madeira (R$)** – Calculado a partir da quantidade (m^3) e preço por m^3. Poderíamos usar fórmula se os dados estiverem acessíveis: por exemplo, se houver colunas ocultas preenchendo o preço unitário via PROCV do fornecedor ou média de mercado. Um caminho: adicionar na aba Fornecedores uma coluna chave com "Fornecedor-Produto-Condicao" e nas colunas de preço já definidas. Então no pedido: `=SE(D2="";""; D2 * SEERRO(PROCV(A2 & "-" & C2 & "-" & Condicao; Fornecedores!chave:preco; colunaPreco; 0); PROCV(C2; PreçosMadeira!tipo:preco; 2; 0) ))`. Isso procura primeiro preço específico do fornecedor; se não achar, pega da tabela de preços médios. Contudo, essa fórmula seria complexa. No nosso caso, deixamos o script definir o valor. Em planilha pura, outra abordagem é usar **QUERY** ou filtros para extrair o preço mais recente da essência na aba PreçosMadeira. Ex: uma célula auxiliar que usa `=FILTER(PreçosMadeira!B:B; PreçosMadeira!A:A = MAX(PreçosMadeira!A:A))` para pegar o último preço. Em resumo, optamos por calcular via script para maior flexibilidade.
- **Total (R$)** – Soma simples: `=SE(F2="";""; H2 + I2)` assumindo H = Custo Frete e I = Custo Madeira. Colocamos condição para não somar se não houver valores (por exemplo, se o pedido ainda não foi calculado). Essa fórmula pode residir na coluna Total de todos os pedidos, atualizando automaticamente quando Custo Frete/Madeira forem preenchidos. (Novamente, nosso script já insere o total, mas a fórmula garante a correção caso algum valor mude manualmente).

Além dessas, usamos fórmulas de procura em outras abas: - Na aba **Fornecedores**, havia fórmulas para calcular **Quantidade m³** a partir de peças e dimensões (já presentes nos dados importados). Mantivemos essas fórmulas: por exemplo, colunas que calculam *m³ por peça* = `ESPESSURA * LARGURA * COMPRIMENTO / 1000000` e *Quantidade m³ total* = `Quantidade de Peças * m³ por peça`.
- Usamos **PROCV (VLOOKUP)** e **FILTER** em algumas colunas auxiliares para cruzar dados. Por exemplo, poderíamos ter na aba Pedidos colunas para *Cidade Origem* e *UF Origem* preenchidas automaticamente: `=SEERRO(PROCV(Fornecedor; Fornecedores!$A:$C; 2; FALSO);"")` para cidade e similar para UF. Isso facilita se quisermos mostrar essas infos ou usar em query.
- A aba **Aux_Cidades** serve para validação: podemos usar validação de dados (Data Validation) na entrada de cidades para evitar erros de digitação, apontando para o intervalo de cidades válidas dessa aba. Isso não é exatamente fórmula, mas uma configuração importante.

Resumindo, combinamos cálculos via script com fórmulas nativas do Sheets para garantir que todos os campos calculados estejam corretos e atualizados. O item 4 do plano foi, assim, atendido: distância e duração via API, custo frete = distância * tarifa, custo madeira = volume * preço, total = soma, com suporte de fórmulas e/ou scripts conforme adequado.

Dashboard – KPIs, Filtros e Visualizações

Organizamos a aba **Dashboard** para fornecer uma visão rápida e interativa do desempenho logístico. Os elementos principais do dashboard incluem:

- **Seletores de Filtro:** No topo do dashboard, inserimos controles de filtro:
- **Filtro de Data:** Dois campos de data (Data Início e Data Fim) ou um intervalo selecionável. O usuário pode definir o período de interesse. As fórmulas no dashboard usam esses valores para filtrar os dados de Pedidos. Por exemplo, usamos a função `QUERY` referenciando as células de data:

```
=QUERY(Pedidos!A:J;
   "select count(A), sum(I), sum(J)
    where A >= date '" & TEXT(DataInicio;"yyyy-MM-dd") & "'
      and A <= date '" & TEXT(DataFim;"yyyy-MM-dd") & "'";
   0)
```

Isto contaria número de pedidos (coluna A supondo Data do pedido), soma do Custo Madeira (coluna I) e soma do Total (coluna J) no intervalo.

- **Filtro de Região:** Uma lista suspensa (Data Validation) para selecionar a região ou estado. Por exemplo, “Todos”, “SP”, “PR”, “SC”, etc., dependendo de onde operam. Essa informação de região pode se referir ao estado de origem do fornecedor ou do destino do pedido. Implementamos como estado de **origem** (fornecedor). Criamos uma lista única de UFs dos fornecedores e colocamos na validação. Ao selecionar, uma célula (digamos B2) guarda o UF escolhido. As fórmulas `QUERY` então incluem algo como `and C = 'SP'` se B2 = "SP", ou ignoram filtro se B2 = "Todos". Isso pode ser feito construindo dinamicamente a string da query ou usando SE:

```
=QUERY(Pedidos!A:J;
   "select sum(J) where " &
   IF(B2="Todos"; "J is not null"; "C = '" & B2 & "'") &
   " group by C"; 0)
```

(Exemplo: soma total de custos filtrado por UF do fornecedor). Para facilitar, também podemos ter colunas auxiliares em Pedidos que já categorizem as regiões (Sudeste, Sul etc.) e filtrar por elas.

- **Filtro de Produto:** Outra lista suspensa para escolher entre Pinus, Eucalipto, etc. Semelhante ao filtro de região, usado nas queries para condicionar o tipo de madeira.

- **Indicadores (KPIs):** Abaixo dos filtros, exibimos cartões com indicadores-chave, calculados em tempo real usando as fórmulas `QUERY` sobre a aba Pedidos filtrada:

- **Total de Pedidos** no período/segmento selecionado (`count()` de linhas).
- **Volume Total (m³)** transportado (`sum(Quantidade_m3)`).
- **Custo Total** (soma de coluna Total R$).
- **Custo Médio por Pedido** (custo total / número de pedidos).
- **Distância Média por Pedido** (média da coluna Distância).
- **Frete Médio por km** (talvez comparar custo frete total / distância total).
- **% Média do Frete no Custo** (por exemplo, percentual do custo total que foi frete vs madeira).

Cada KPI pode ser apresentado em um pequeno bloco. Usamos formatação visual (números com separador, R$, e talvez cores).

- **Gráficos/Sparklines:** Incorporamos visualizações para tendências:
- **Preço da Madeira:** Gráfico de linha ou sparkline mostrando a variação de preço do Pinus e Eucalipto ao longo do último mês. Implementação: duas mini sparklines ou um gráfico combinando duas séries. Exemplo de uso do `SPARKLINE`:
  - Célula com fórmula: `=SPARKLINE(OFFSET(PreçosMadeira!$B$2; COUNTA(PreçosMadeira!$B:$B)-30; 0; 30; 1); {"linewidth", 2\;"color","green"})` para Pinus (pega os últimos 30 registros da coluna B – preço Pinus). E similar para Eucalipto com cor diferente. Isso mostra a tendência de preço recente diretamente no dashboard. Alternativamente, um gráfico line chart convencional seria mais visível, se permitido no formato final.
- **Volume Transportado vs. Meta:** Se houver uma meta mensal de volume, podemos usar um gráfico de barras mostrando volume por mês vs meta. Esse dado pode ser derivado via QUERY agrupando Pedidos por mês. Por exemplo, usar `QUERY(... group by month A ...)` ou simplesmente preparar uma tabela pivô. Para ilustrar, fizemos um gráfico de colunas para volumes dos últimos 6 meses.
- **Distribuição por Região/Produto:** Podemos inserir gráficos de pizza ou coluna mostrando participação de cada região no total de volume, ou de cada tipo de produto nas vendas. Isso pode ser alimentado por fórmulas somando valores por categoria. Ex: usar `QUERY` para somar total por produto: `=QUERY(Pedidos!C:J; "select C, sum(J) group by C";0)` retornaria cada produto e soma do total R$. O dashboard então apresenta um gráfico de pizza usando esse range (produto vs receita).

- **Indicador de Tendência:** Poderíamos usar um **SPARKLINE de tendência semanal** de pedidos: criar uma pequena tabela auxiliar no dashboard com quantidade de pedidos por semana nas últimas N semanas (usando QUERY ou CONT.SE com intervalo de datas) e passar isso a um SPARKLINE (charttype "column"). Isso daria uma ideia se os pedidos estão aumentando ou diminuindo.

- **Layout:** Organizamos o Dashboard de forma limpa: os filtros no topo (talvez em células com fundo cinza claro e texto de destaque), os KPIs logo abaixo em caixas coloridas (podemos mesclar células para criar caixas maiores e usar formatação condicional para mudar cor conforme valores ou simplesmente fixar uma cor tema). Os gráficos colocamos mais embaixo ou à direita, para não poluir a visão inicial.

- **Interatividade:** Graças ao uso de fórmulas `QUERY` reagindo aos valores dos filtros, o Dashboard se atualiza automaticamente quando o usuário muda, por exemplo, o produto ou a data selecionada. Usamos também a função `INDIRETO` em algumas fórmulas para flexibilidade. Exemplo: definir o intervalo base de dados do QUERY via Indireto, caso a posição dos dados varie.

Em resumo, o Dashboard permite à equipe analisar rapidamente: - Qual foi o custo total de frete e madeira em um período e região específicos. - Qual produto teve maior volume ou receita. - Tendência dos custos de matéria-prima (componente crucial na precificação). - Desempenho relativo de fornecedores ou regiões (poderíamos até listar top 5 fornecedores por volume, usando QUERY com `order by` e `limit 5`).

Essas informações visuais auxiliam em **decisões B2B rápidas e precisas**, atendendo ao foco do projeto: permitir otimização logística (escolha de fornecedor ideal, negociação de fretes) com base em dados atualizados.

Documentação e Guia de Uso (README)

Para garantir que a equipe aproveite todos os recursos, incluímos um **guia rápido de uso** integrado à planilha:

- **Aba README:** A última aba da planilha é dedicada a instruções. Nela escrevemos em formato de texto os objetivos da planilha, explicação de cada aba e passo-a-passo para usar as automações. Por exemplo, instruímos que para **calcular um novo pedido**, o usuário deve preencher os dados nas colunas adequadas na aba Pedidos e então escrever "Calcular" na coluna Status daquele pedido – em poucos instantes os campos serão preenchidos automaticamente. Explicamos também que os dados de preços e fretes são atualizados diariamente de forma automática (mas podem ser acionados manualmente pelo menu), portanto os usuários não devem editar manualmente essas abas, apenas consultar.
- No README também detalhamos quaisquer premissas ou simplificações do modelo (por ex.: “assumimos que o preço informado pelos fornecedores já é por m^3 de madeira seca; caso seja madeira verde, considere um desconto X%...” ou “o custo do frete é calculado supondo transporte exclusivo por caminhão do tipo selecionado, não incluindo eventuais transbordos”).
- **Guia via Menu:** Como mostrado, ao clicar em **“Ver Guia de Uso”** no menu personalizado, o usuário recebe um pop-up com as principais dicas. Isso é útil para relembrar rapidamente como operar, sem precisar abrir a aba README. O texto do alerta enfatiza os pontos-chave:
- Adicionar/editar pedidos e acionar o cálculo.
- Horários das atualizações automáticas.
- Uso do Dashboard (seleção de filtros).

- Papel do log (somente monitoramento, geralmente não precisa de ação do usuário, exceto consultar se algo deu errado).

- **Comentários e Proteções:** Adicionamos comentários em células importantes para explicar cálculos (ex.: na primeira célula de Custo Frete pode ter um comentário "Calculado automaticamente = Distância * Valor por km do frete"). Além disso, recomendamos **proteger** as abas de referência (*Fornecedores, Caminhões, Fretes, PreçosMadeira, Aux_Cidades, Logs*) para que a equipe não altere acidentalmente fórmulas ou dados base. Apenas a aba *Pedidos* e possivelmente inputs no *Dashboard* precisam ser editáveis pelos usuários comuns. Documentamos essa recomendação no README.

Com essa documentação integrada, a equipe poderá compreender e confiar na ferramenta. **Resumo do guia para a equipe:**

- As abas de dados mestre (Fornecedores, etc.) já estão preenchidas e padronizadas – mantenha-as atualizadas se houver novas informações de fornecedor ou mudanças (ex.: novo preço negociado diretamente com algum fornecedor específico, você pode editar na aba Fornecedores).
- Para registrar um novo pedido de compra/venda de madeira, vá na aba **Pedidos** e preencha uma nova linha com *Fornecedor*, *Destino* (cidade de entrega), *Produto* e *Quantidade*. Depois, na coluna **Status** selecione "Calcular". Em segundos, o sistema preencherá distância, duração estimada do transporte, calculará o custo de frete (com base no tipo de caminhão adequado) e o

custo da madeira (com base no preço mais recente), resultando no **Total** estimado daquele pedido.

- Veja na aba **Dashboard** os efeitos dos pedidos: por exemplo, se você filtrar por data do mês atual, verá quantos pedidos já estão programados, qual o custo total previsto e outras métricas. Use os filtros de **Região** e **Produto** para focar em determinados mercados (por exemplo, "Quanto gastamos com Pinus no Sul do país este trimestre?").
- As atualizações de preços de madeira acontecem todo dia às 6h automaticamente. As de frete às 6h30. Caso você abra a planilha e queira os dados mais frescos manualmente, use o menu **Logística > Atualizar Preços** ou **Atualizar Tabela de Frete**.
- A aba **Logs** mostrará registros de quando essas atualizações ocorreram ou se houve algum erro de conexão com APIs. Se algo não foi atualizado, verifique o log para diagnosticar (por exemplo, pode indicar "ERRO na atualização de preços: conexão recusada"). Os logs são apagados semanalmente para não crescer demais.
- Em caso de dúvidas, consulte o README ou clique em **Logística > Ver Guia de Uso** para este resumo. Mantenha a estrutura das abas – não renomeie ou delete abas, pois isso desconectaria os scripts.

Conclusão: A planilha Google Sheets implementada oferece uma solução integrada para automação da logística de madeira B2B, combinando **precisão** (dados atualizados de distância, tempo, preços e fretes), **rapidez** (cálculos instantâneos via Apps Script) e **integração com APIs** confiáveis (Google Maps para rotas, Tollguru para pedágios futuramente, fontes de mercado para preços de commodities e frete). A equipe poderá tomar decisões informadas – como qual fornecedor é mais competitivo entregue em determinado destino – de forma quase imediata, economizando tempo e reduzindo erros. Todas as etapas do plano detalhado foram implementadas conforme solicitado, com documentação interna para facilitar a adoção pela equipe usuária. Boa utilização!
