**Documento de Design de Software (DDS)**

**Projeto:** Sistema de Chamada de Veículos (SCV)

**Versão:** 2.1 – Build de Produção Estável

**Data da Última Revisão:** 13 de setembro de 2025

**Autores:** Hudson Rodrigues

**1. Visão Geral e Arquitetura**

**1.1. Introdução**

Este documento detalha a arquitetura final e o funcionamento interno do Sistema de Chamada de Veículos (SCV), uma aplicação web full-stack, reativa e projetada para operar em tempo real. O sistema substitui o fluxo de trabalho manual de convocação de veículos por um processo digital, rastreável e otimizado, servindo como a principal ferramenta para a gestão de pátio.

**1.2. Arquitetura em Três Camadas**

A arquitetura do sistema é baseada num modelo de três camadas para garantir a separação de responsabilidades (Separation of Concerns), escalabilidade e manutenibilidade.

* **Camada de Apresentação (Frontend):** Composta por interfaces de utilizador leves e reativas (analista.html, motorista.html) construídas com HTML, CSS e JavaScript puro. A comunicação com o backend é realizada exclusivamente através de WebSockets (Socket.IO) para garantir a instantaneidade dos dados.
* **Camada de Lógica de Negócio (Backend):** O núcleo da aplicação, construído em **Node.js** com o framework **Express**. As suas responsabilidades são gerir o ciclo de vida das conexões WebSocket, orquestrar os fluxos de negócio (eventos de check-in, chamada, finalização), e atuar como a única ponte entre os clientes e o banco de dados.
* **Camada de Dados (Persistência):** Um servidor de banco de dados **MySQL** é utilizado para armazenar de forma permanente todo o estado da aplicação. Isto inclui a fila de veículos, os seus status e as métricas de tempo, garantindo a integridade e a durabilidade dos dados.

**2. Estrutura do Projeto**

A base de código está organizada de forma modular para refletir a arquitetura em camadas.

* /db/ - **Camada de Acesso a Dados (DAL)**
  + connection.js: Responsável por configurar e exportar o pool de conexões com o MySQL.
  + queries.js: Abstrai toda a lógica SQL em funções assíncronas reutilizáveis. Este é o único local que contém código SQL.
* /public/ - **Ficheiros do Frontend**
  + analista.html: Interface do painel de controlo.
  + motorista.html: Interface de check-in e notificação.
  + style.css: Folha de estilos centralizada.
* server.js - **Ponto de Entrada e Orquestrador**
  + Ficheiro principal que inicia a aplicação, gere o servidor web e orquestra os eventos entre o frontend e a camada de dados.
* schema.sql - **Definição da Estrutura de Dados (DDL)**
  + Script SQL para a criação e definição da tabela veiculos.
* .env – **Configuração de Ambiente**
  + Ficheiro de configuração para armazenar dados sensíveis (credenciais da base de dados, porta do servidor, etc.).

**3. Esquema do Banco de Dados (veiculos)**

A persistência de dados é centralizada na tabela veiculos.

**3.1. Dicionário de Dados**

| Coluna | Tipo de Dado | Restrições | Descrição Funcional |
| --- | --- | --- | --- |
| id | INT | PK, AI | Identificador técnico, único e sequencial. |
| placa | VARCHAR(10) | NOT NULL, UNIQUE | Placa do veículo. Chave de negócio principal. |
| motorista | VARCHAR(255) | NOT NULL | Nome do motorista. |
| transportadora | VARCHAR(255) | NOT NULL | Nome da empresa transportadora. |
| chegada | DATETIME | NOT NULL | Timestamp exato do momento do check-in. |
| status | VARCHAR(50) | NOT NULL | Estado atual do veículo no fluxo (Aguardando, Chamado, Acessando Pátio, Finalizado). |
| socket\_id | VARCHAR(255) | NULL | ID técnico da conexão WebSocket do motorista, para envio de notificações diretas. |
| horario\_chamada | DATETIME | NULL | Timestamp de quando o analista acionou a chamada. Usado para "congelar" o cálculo do tempo de espera. |
| tempo\_espera\_minutos | INT | NULL | **Métrica de KPI:** Armazena o tempo total de espera em minutos, calculado e gravado no momento da chamada. |

Exportar para as Planilhas

**4. Fluxo de Dados e Lógica de Negócio**

**4.1. Check-in do Veículo**

1. **Frontend (motorista.html):** O utilizador preenche o formulário. O JavaScript força os caracteres para toUpperCase() e emite o evento motorista:checkin.
2. **Backend (server.js):** Recebe o evento e invoca db.addOrUpdateVehicle(), passando os dados do formulário e o socket.id da conexão.
3. **Camada de Dados (db/queries.js):** A função addOrUpdateVehicle utiliza uma query INSERT ... ON DUPLICATE KEY UPDATE. Esta abordagem trata um novo check-in de uma placa já existente como uma reentrada na fila, atualizando os dados e redefinindo os campos de tempo para NULL.
4. **Broadcast (server.js):** O servidor emite lista:atualizada para todos os clientes conectados.

**4.2. Chamada do Veículo e Cálculo de KPI**

1. **Frontend (analista.html):** O analista clica em "Chamar". O evento analista:chamar é emitido com a placa.
2. **Backend (server.js):** a. Recebe o evento e armazena a hora atual (horarioChamada). b. Invoca db.getVehicleByPlaca() para obter o horário de chegada do veículo. c. **Cálculo de KPI:** Calcula a diferença entre horarioChamada e chegada, convertendo-a para minutos (tempoEsperaMinutos). d. Invoca db.updateVehicleCall(), passando a placa, o novo status "Chamado", o horarioChamada e o tempoEsperaMinutos calculado.
3. **Camada de Dados (db/queries.js):** A função updateVehicleCall executa uma única query UPDATE que atualiza atomicamente as três colunas (status, horario\_chamada, tempo\_espera\_minutos).
4. **Notificação e Broadcast (server.js):** O servidor envia um evento veiculo:chamado direcionado ao socket\_id do motorista e, em seguida, emite lista:atualizada para todos os clientes.

**4.3. Renderização do Painel do Analista**

1. **Frontend (analista.html):** Ao receber o evento lista:atualizada, o script armazena os dados e inicia um setInterval de 1 segundo (1000ms).
2. **Lógica de Renderização:** A cada segundo, a função renderizarTabela é executada e, para cada veículo, aplica a seguinte regra para o tempo de espera:
   * **Se o status for "Aguardando":** O tempo é calculado dinamicamente como (Tempo Atual - Tempo de Chegada) e exibido no formato minutos:segundos. O cálculo Math.max(0, ...) previne um bug visual de tempo negativo.
   * **Se o status NÃO for "Aguardando" (e horario\_chamada existir):** O tempo é calculado de forma fixa como (horario\_chamada - chegada), exibido em minutos e marcado como "(Final)". Esta regra garante que, uma vez chamado, o tempo de espera fica permanentemente congelado, mesmo que o status mude posteriormente para "Acessando Pátio".

**5. Histórico de Revisões da Versão 2.1**

* **Funcionalidade:** Adicionado o campo tempo\_espera\_minutos à base de dados. O sistema agora calcula e armazena permanentemente o tempo de espera de cada veículo no momento da chamada, para futuras análises de KPI.
* **Melhoria de UI:** O timer de espera no painel do analista foi ajustado para atualizar a cada segundo (setInterval de 1000ms), proporcionando uma experiência de monitorização mais fluida.
* **Correção de Bug:** A lógica de renderização no analista.html foi robustecida para garantir que o tempo de espera permaneça congelado após a chamada, mesmo quando o status do veículo muda para "Acessando Pátio".