Licenciatura em Engenharia e Gestão de Sistemas de Informação

Universidade do Minho

# Fundamentos dos Sistemas Distribuídos

2025/2026

## Gestor de Parques de Estacionamento

Grupo 3 PL3


Afonso Leite

a108552

@uminho.pt



João Ribeiro

a108545

@uminho.pt



José Silva

a104070

@uminho.pt

# Implementação

## Arquitetura Geral

A Fase 3 introduz um novo componente no sistema distribuído desenvolvido anteriormente: o Cliente Web, responsável por consolidar e apresentar ao utilizador final a informação recolhida pelo Gestor de Parques e disponibilizada pelos Parques de Estacionamento.
Este cliente funciona como uma camada de visualização e interação, suportada por uma API interna e uma interface web dinâmica, totalmente desenvolvida em Flask.

A arquitetura geral desta fase assenta em três entidades já existentes – Gestor de Parques, Parques de Estacionamento e Simuladores de Lugares – e adiciona-lhes o novo elemento, o Cliente Web, que estabelece comunicações HTTP com o Gestor e com cada parque.
Assim, o Cliente Web assume o papel de agregador de informação, recolhendo a lista de parques ativos diretamente do Gestor e consultando os dados detalhados de cada parque (lotação, ocupação, tarifas, localização e custo simulado) através da respetiva API REST.

Do ponto de vista arquitetural, o Cliente Web divide-se em duas camadas interdependentes:

- API interna (backend): responsável por contactar o Gestor e os parques, tratar respostas e uniformizar dados antes de os fornecer ao frontend.

- Interface Web (frontend): construída em HTML, CSS e JavaScript, gerada e servida dinamicamente pelo Flask, sendo responsável por apresentar a informação ao utilizador e permitir interação.

Toda a arquitetura mantém compatibilidade com as fases anteriores, reutilizando os mecanismos de comunicação já implementados, sem modificar a infraestrutura dos parques ou dos simuladores. O Cliente Web estende assim o sistema para uma dimensão orientada ao utilizador, mantendo coerência com o modelo distribuído.

# Estrutura dos Módulos

A Fase 3 introduz essencialmente um novo módulo: cliente_web.py, que contém toda a lógica do Cliente Web.
Os restantes módulos herdam a estrutura previamente desenvolvida, não sendo alterados de forma substancial nesta fase.

A nova estrutura inclui cliente_web.py:

Contém:

- Configurações do Gestor (endereço e porta).
- Implementação da API interna:
    - /api/parques – obtém lista de parques ativos.
    - /api/info – obtém dados detalhados via /info.
    - /api/custo – obtém custo via /custo?tempo=X.
- Geração da interface gráfica em HTML/CSS/JS.
- Funções JavaScript integradas para atualização, seleção de parques, cálculo de custos e apresentação gráfica da ocupação.

A API interna funciona como camada de abstração, garantindo que o frontend recebe dados uniformizados, mesmo quando contacta parques de outros grupos com campos inconsistentes.

Os módulos parque.py, config.py, protocolo.py, lugar.py
Mantêm a lógica implementada anteriormente e continuam a fornecer os endpoints REST usados pelo Cliente Web.Nesta fase, nenhum destes módulos é modificado, mas o seu papel torna-se fundamental porque o Cliente Web depende deles para recolher informação fiável sobre cada parque.

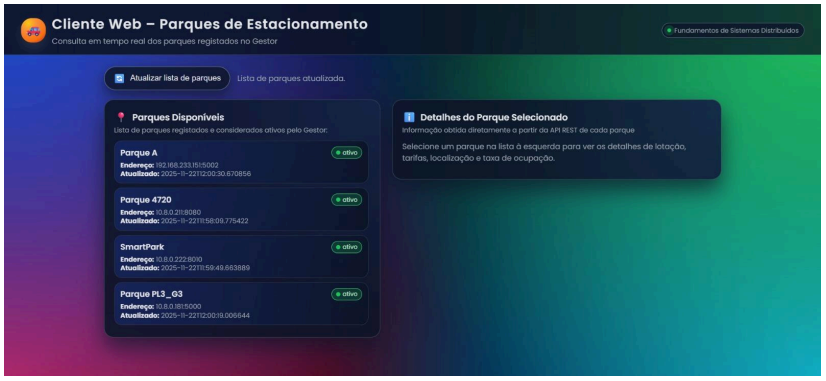


Figura 1 - A interface Web apresenta de forma clara a informação devolvida pelo endpoint /info: lotação total, lugares livres, tarifas, coordenadas geográficas e taxa de ocupação. A barra colorida representa visualmente a percentagem de ocupação em tempo real.


Figura 2 - A aplicação permite ainda estimar o custo de estacionamento, enviando um pedido ao endpoint /custo?tempo=X. O utilizador introduz o tempo desejado e o Cliente Web atualiza o resultado de forma dinâmica.

# Simulador de Lugares e Protocolo de Comunicação

Embora esta fase do projeto não introduza alterações diretas no simulador de lugares nem no protocolo TCP, ambos permanecem essenciais para o funcionamento global do sistema.

O simulador continua a gerar estados LIVRE e OCUPADO e a transmiti-los ao parque via TCP.
O protocolo mantém as mensagens INIT, UPDATE, INFO e ERRO, responsáveis pela comunicação estruturada entre os lugares e o Parque de Estacionamento.

A principal diferença nesta fase é que os dados transmitidos pelo simulador são agora indiretamente utilizados pelo Cliente Web, que os obtém através da API REST dos parques.
Assim, embora não haja alterações técnicas na implementação, há uma mudança funcional: os dados do protocolo TCP passam a influenciar diretamente a informação visualizada pelo utilizador final.

# Parque de Estacionamento e Gestor de Parques

## 4.1 Papel do Gestor de Parques

O Gestor mantém o seu papel de entidade centralizadora, responsável por receber registos periódicos enviados pelos parques.
Durante a Fase 3, este componente torna-se a principal fonte de informação do Cliente Web, que consulta o endpoint:

GET /parque

para obter a lista atualizada de parques ativos.

### 4.2 Papel dos Parques de Estacionamento

Cada parque continua a disponibilizar a sua API REST, composta por endpoints como:

- /info
- /ocupacao
- /custo
- /lugares
- /health

Estes endpoints são agora consumidos diretamente pelo Cliente Web, que recolhe e processa a informação para apresentá-la ao utilizador. Assim, o Parque de Estacionamento torna-se um fornecedor de dados para uma nova entidade (o Cliente Web), reforçando o caráter distribuído do sistema.

## Concorrência e sincronização

Embora o Cliente Web não adote um modelo intensivo de multithreading como as fases anteriores, existem três aspetos de concorrência que influenciam esta fase:

### 5.1 Concorrência no lado do Cliente Web (Flask)

O Flask lida com múltiplos pedidos simultâneos vindos do navegador, fazendo com que:

- a lista de parques possa ser atualizada enquanto o utilizador consulta detalhes;
- o cálculo de custo possa ocorrer simultaneamente com consultas ao Gestor.

Esta concorrência é gerida pelo servidor web do Flask.

### 5.2 Concorrência entre entidades distribuídas

O Cliente Web comunica com entidades que também operam concorrencialmente:

- o Gestor recebe registos periódicos dos parques;
- os parques recebem mensagens TCP dos simuladores;
- os parques respondem a pedidos REST.

Assim, a concorrência torna-se distribuída, envolvendo múltiplos processos independentes.

### 5.3 Sincronização funcional através de refresh automático

A interface Web aplica:

- atualização automática a cada 30 segundos;
- carregamento inicial assíncrono;
- atualização independente dos detalhes e do custo.

Deste modo, mesmo sem bloqueios explícitos, garante-se que a interface espelha o estado atual do sistema de forma coerente e sincronizada.

## Considerações finais

A Fase 3 completou o sistema distribuído desenvolvido ao longo do projeto, introduzindo um Cliente Web capaz de consultar o Gestor de Parques e apresentar ao utilizador informação detalhada sobre cada parque. Esta interface acrescentou uma camada essencial de visualização, tornando o sistema mais acessível e evidenciando a integração entre os componentes criados nas fases anteriores.

A solução implementada mostrou-se funcional e estável, permitindo visualizar parques ativos, consultar o seu estado e simular custos de estacionamento. A API interna do Cliente Web garante a compatibilidade com parques de diferentes grupos, assegurando robustez mesmo perante formatos inconsistentes ou falhas de comunicação.

Do ponto de vista técnico, esta fase reforçou a importância da comunicação HTTP, da abstração entre componentes e da tolerância a falhas em sistemas distribuídos. A interação entre Gestor, Parques e Cliente Web revelou-se coerente e eficaz, consolidando a arquitetura global do projeto.

Em conjunto, os resultados obtidos demonstram que o sistema alcançou maturidade e cumpre plenamente os objetivos definidos, integrando comunicação TCP, serviços REST e visualização web num ambiente distribuído coeso e extensível.