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

Nesta fase, o sistema foi ampliado para além da simples gestão de lugares via TCP, passando a integrar dois novos componentes fundamentais: o registo automático no Gestor de Parques e a disponibilização de uma API REST pública.

O Parque de Estacionamento mantém-se como um servidor TCP, responsável por receber e processar as mensagens provenientes dos lugares simulados. Contudo, passou também a comunicar periodicamente com o Gestor de Parques, através de pedidos HTTP, e a disponibilizar serviços externos acessíveis a aplicações de terceiros.

O registo no Gestor é efetuado por uma thread dedicada, responsável por enviar, a cada três minutos, um pedido HTTP POST que inclui o nome, o endereço IP e a porta do Parque. Este mecanismo garante a descoberta automática do Parque e a sua manutenção como entidade ativa no Gestor, permitindo que outros clientes e sistemas o identifiquem em tempo real.

Em paralelo, o Parque disponibiliza uma API REST, implementada com o framework Flask, que fornece informação em tempo real sobre o seu estado e permite a simulação de custos de estadia. A API inclui vários endpoints, abrangendo desde a consulta de dados básicos até à obtenção de métricas de ocupação e indicadores de monitorização técnica. Esta funcionalidade consolida o Parque como uma entidade distribuída completa, preparada para interagir com aplicações externas e futuras interfaces web.

Todos os componentes — o servidor TCP, a thread de registo no Gestor e a API REST — executam em linhas de execução independentes, garantindo que o sistema se mantém responsivo e capaz de processar múltiplos pedidos em simultâneo.

Estrutura dos Módulos

A estrutura global do projeto manteve-se modular e coerente, preservando a separação de responsabilidades entre componentes.

Os módulos config.py, protocolo.py e lugar.py mantiveram as suas funções originais, não tendo sofrido alterações significativas nesta fase. O desenvolvimento desta fase concentrou-se no módulo parque.py, que foi substancialmente expandido para incorporar as novas funcionalidades:

- Registo automático no Gestor de Parques — efetuado através de uma thread dedicada, responsável por enviar periodicamente (a cada três minutos) os dados identificativos do parque, incluindo o nome, o endereço IP e a porta da API REST.
- API REST pública — desenvolvida com o framework Flask, disponibiliza múltiplos endpoints para consulta de informação, cálculo de custos de estadia, monitorização técnica e visualização através de um dashboard HTML.
- Gestão de reconexões de lugares — permite a reatribuição automática dos mesmos identificadores (IDs) após falhas de ligação, garantindo a persistência e a consistência dos dados provenientes dos sensores simulados.
- Sistema de registo de eventos — as principais operações e mensagens do servidor são impressas na consola através da função log(), que apresenta informação contextualizada sobre o estado do sistema e as comunicações realizadas.

Estas alterações consolidaram o Parque de Estacionamento como um componente autónomo, integrado e robusto, preparado para operar em conjunto com o Gestor de Parques e com clientes externos, sem comprometer a simplicidade e a modularidade da arquitetura base.

Simulador de Lugares e Protocolo de Comunicação

O simulador de lugares e o protocolo de comunicação mantêm a estrutura definida na Fase 1, baseando-se em sockets TCP e em mensagens de texto no formato COMANDO;;param1=valor1;;param2=valor2.

Cada lugar é representado por uma thread independente, que comunica periodicamente com o Parque, alterando o seu estado entre LIVRE e OCUPADO de acordo com as probabilidades configuradas. A principal melhoria introduzida nesta fase prende-se com a gestão de reconexões: após uma falha de ligação, o lugar volta a contactar o Parque e recupera automaticamente o mesmo identificador (ID) que lhe havia sido previamente atribuído. Este comportamento garante a persistência e a consistência dos dados, mesmo em cenários de interrupção temporária.

O protocolo mantém os comandos fundamentais — INIT, UPDATE e INFO — bem como o mecanismo de validação de erros (ERRO), assegurando compatibilidade total com a versão anterior e uma comunicação fiável entre todas as entidades do sistema.

Parque de Estacionamento e Gestor de Parques

Nesta segunda fase, o Parque de Estacionamento evoluiu de um simples servidor TCP para uma entidade distribuída completa, capaz de comunicar com o Gestor de Parques e de expor uma API REST pública.

O registo no Gestor é efetuado por uma thread dedicada, que, a cada três minutos, envia um pedido HTTP POST para o serviço central disponibilizado pelos docentes, no endereço http://192.168.233.151:6666/parque.

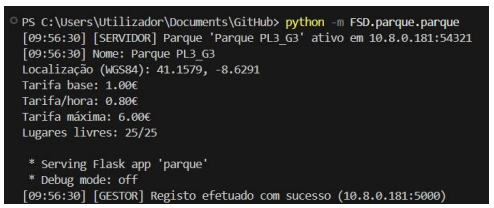


Figura 1 - Registo periódico do Parque de Estacionamento no Gestor, comprovando a comunicação HTTP bem-sucedida.

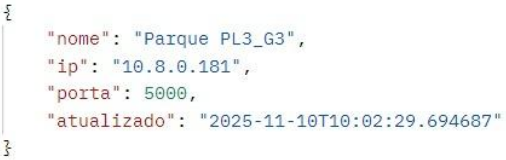


Figura 2 – Resposta do Gestor de Parques ao pedido de registo, com os dados identificativos e a hora da última atualização.

O corpo da mensagem inclui o nome do parque, o endereço IP (obtido dinamicamente através da interface VPN) e a porta onde a API REST se encontra ativa.

Este mecanismo assegura que o parque é descoberto automaticamente e mantido na lista de parques ativos, permitindo a sua consulta por aplicações externas e pelos restantes grupos.

Em paralelo, o Parque disponibiliza uma API REST, implementada com o framework Flask, que fornece acesso em tempo real ao estado do sistema. Os endpoints desenvolvidos incluem funcionalidades de consulta de informação, cálculo de custos de estadia e monitorização técnica, nomeadamente:

- /info – devolve os dados gerais do parque, incluindo nome, capacidade, lugares livres e tarifas;

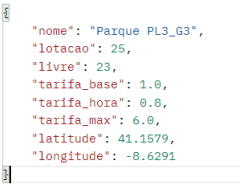
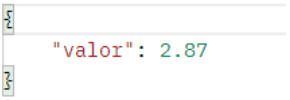


Figura 3 – Resposta do endpoint /info, com os principais dados do parque e respetivos parâmetros de configuração.

- /custo?tempo=X – calcula o custo estimado para um tempo de permanência específico;



Foi usado o valor de X = 140.

Figura 4 – Resposta do endpoint /custo, ilustrando o cálculo do custo estimado de estadia em função do tempo de permanência fornecido.

- /ocupacao – apresenta a taxa de ocupação e o número de lugares livres e ocupados;

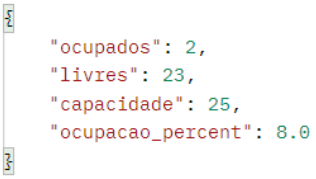


Figura 5 – Resposta do endpoint /ocupacao, com a taxa de ocupação atual do parque e o número de lugares livres e ocupados.

- /lugares – lista todos os lugares e o respetivo estado atual;

Figura 6 – Resposta do endpoint /lugares, listando todos os lugares existentes e o respetivo estado.

- /health – fornece dados técnicos de monitorização, como o tempo de atividade e o estado de sincronização com o Gestor;

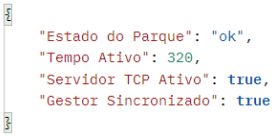


Figura 7 – Resposta do endpoint /health, contendo indicadores técnicos de monitorização.

- /dashboard – apresenta uma interface HTML simples, com informação visual do parque e atualização automática.

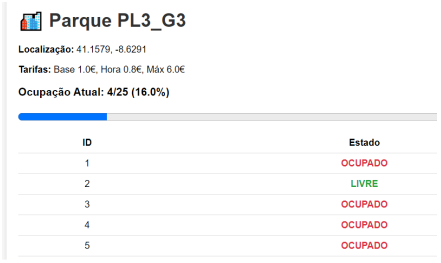


Figura 8 – Interface HTML disponibilizada pelo endpoint /dashboard, apresentando visualmente a ocupação do parque e o estado individual dos lugares

As mensagens de operação e de depuração são apresentadas em consola através da função log(), ativada quando LOG_VERBOSO = True.

Este sistema permite acompanhar em tempo real o comportamento do parque, os pedidos REST recebidos e as comunicações efetuadas com o Gestor.

Concorrência e sincronização

A execução concorrente constitui um elemento central nesta fase do projeto. O sistema realiza, em simultâneo, comunicações TCP com os lugares, pedidos HTTP ao Gestor e operações REST com clientes externos.

Cada uma destas tarefas é gerida numa thread independente, assegurando paralelismo e evitando bloqueios entre componentes. Para garantir a integridade dos dados — especialmente nas operações que envolvem a atualização de estados e contagens — é utilizado um mecanismo de sincronização através da classe threading.Lock(). Este bloqueio impede que múltiplas threads alterem simultaneamente os mesmos dados, prevenindo condições de corrida e inconsistências no estado do sistema.

Durante os testes realizados, o sistema demonstrou estabilidade e previsibilidade sob carga concorrente. Mesmo com múltiplas ligações simultâneas de lugares e pedidos REST em paralelo, o Parque manteve tempos de resposta reduzidos e um comportamento estável, comprovando a eficácia da abordagem de concorrência implementada.

Considerações finais

A Fase 2 consolidou o Parque de Estacionamento como um sistema distribuído completo, interligando as várias componentes desenvolvidas e estabelecendo a base para futuras expansões. A integração com o Gestor de Parques e a disponibilização da API REST pública representaram os principais avanços desta etapa, reforçando a comunicação, a escalabilidade e a capacidade de monitorização do sistema.

A manutenção de uma arquitetura modular e concorrente permitiu evoluir o projeto de forma controlada e extensível. As novas funcionalidades — registo automático, reconexão de lugares e registo de eventos em consola — contribuíram para uma maior fiabilidade e para um funcionamento mais próximo de um sistema real.

Os resultados obtidos demonstram a maturidade do sistema e a adequação das soluções técnicas adotadas. A experiência adquirida nesta fase reforça a compreensão dos princípios dos sistemas distribuídos e constitui uma base sólida para a Fase 3, dedicada à integração com clientes web e à introdução de mecanismos de segurança e autenticação.