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

O sistema desenvolvido adota uma arquitetura cliente-servidor baseada em sockets TCP, assegurando uma comunicação bidirecional fiável entre os componentes.
O servidor representa o Parque de Estacionamento, sendo responsável pela gestão global dos lugares e respetivos estados. Os clientes correspondem aos Lugares, que são simulados através de threads independentes, permitindo a comunicação periódica e concorrente com o Parque. O protocolo de comunicação, definido pelo grupo, baseia-se em mensagens de texto delimitadas por ";;", possibilitando pedidos simples de leitura e atualização de estado. As principais mensagens implementadas são:

- INIT – pedido inicial de registo de um novo lugar,
- UPDATE – envio do estado atual do lugar;
- INFO – pedido de resumo informativo do parque.

O servidor responde sempre com um comando OK (sucesso) ou ERRO (falha), acompanhado dos parâmetros relevantes à operação.

Estrutura dos Módulos

O projeto está dividido em quatro ficheiros principais:

Módulo	**Responsabilidade principal**
config.py	Define as constantes globais de configuração do sistema (HOST, PORT, CAPACIDADE, probabilidades PO/PL, intervalos de simulação, etc.).
protocolo.py	Implementa o protocolo comum de comunicação entre o servidor e os clientes, disponibilizando as funções codificar() e descodificar(), bem como as exceções associadas à validação de mensagens.
parque.py	Contém a classe Parque e o servidor TCP, responsável por gerir os lugares e processar os pedidos provenientes dos clientes.
lugar.py	Simula os lugares de estacionamento, através de threads que comunicam periodicamente com o servidor, representando os sensores do sistema.

Parque de Estacionamento

A classe Parque encapsula toda a informação e lógica associada à gestão de um parque de estacionamento.
Atributos principais:

- Nome – identificação textual do parque;
- Localização – coordenadas geográficas em formato WGS84;
- Tarifas – valores base, por hora e máxima;
- Capacidade – número total de lugares disponíveis;
- Lugares – dicionário que armazena o estado de cada lugar (LIVRE ou OCUPADO).

Métodos principais:

- registar_lugar() – atribui um identificador único e adiciona o novo lugar como LIVRE;
- atualizar_estado() – atualiza o estado de um lugar previamente registado;
- contar_ocupados() – contabiliza o número de lugares atualmente OCUPADOS;
- info() – gera um resumo textual com a informação global do parque.

O servidor TCP cria uma instância da classe Parque e aguarda novas ligações de clientes. Cada ligação recebida (correspondente a um “Lugar”) é tratada numa thread independente através da função handle_client(), garantindo concorrência e isolamento entre os clientes.
As mensagens recebidas são interpretadas pelo método descodificar(), sendo o comportamento do servidor o seguinte:

- INIT → devolve um novo identificador único (ID);
- UPDATE → valida os parâmetros e atualiza o estado do lugar;
- INFO → retorna as informações atuais do parque;
- Mensagens inválidas → devolvem um comando ERRO, com a descrição do problema.

Esta estrutura permite uma gestão eficiente e escalável do parque, servindo de base para a simulação dos lugares descrita na secção seguinte.

Simulador de Lugares

A aplicação Lugar representa os sensores responsáveis por monitorizar cada lugar de estacionamento.
Cada lugar é simulado numa thread independente, conforme indicado no enunciado, permitindo testar simultaneamente a comunicação de vários sensores com o Parque e avaliar o comportamento concorrente do sistema.
O módulo lugar.py executa o seguinte ciclo operacional:

1. Liga-se ao Parque via socket TCP (utilizando o endereço e a porta definidos em config.py);
2. Envia o comando INIT para obter um identificador único (resposta: OK;;id=<n>);
3. A cada intervalo de simulação (INTERVALO_SIMULACAO), calcula o novo estado com base nas probabilidades de ocupação (PO) e libertação (PL);
4. Envia o comando UPDATE com o estado atual do lugar (LIVRE ou OCUPADO);
5. Em cerca de 25 % das mensagens, introduz erros simulados (comando inválido, formato incorreto, parâmetros em falta, etc.) para testar a robustez do protocolo de comunicação.

As funções auxiliares asseguram o envio e receção fiáveis das mensagens, enquanto o método _criar_threads_lugares() lança todas as threads de simulação.
O Parque gere a concorrência através de um bloqueio (Lock), garantindo a consistência dos dados partilhados entre as várias threads.
As figuras seguintes ilustram o registo inicial dos lugares e a atribuição dos seus identificadores únicos.

```
[SERVIDOR]: OK;;id=1
[INFO] Lugar registado com ID 1
[SERVIDOR]: OK;;id=2
[INFO] Lugar registado com ID 2
[SERVIDOR]: OK;;id=3
[INFO] Lugar registado com ID 3
[SERVIDOR]: OK;;id=4
[INFO] Lugar registado com ID 4
[SERVIDOR]: OK;;id=5
[INFO] Lugar registado com ID 5
```

Figura 1 - O terminal do "Lugar" mostra as respostas do servidor com os IDs atribuídos

```
[17:54:17] [+] Ligação estabelecida com ('127.0.0.1', 62792)
[17:54:17] [RECEBIDO de ('127.0.0.1', 62792)] INIT
[17:54:17] [REGISTADO] Lugar 1 criado.
[17:54:18] [+] Ligação estabelecida com ('127.0.0.1', 62793)
[17:54:18] [RECEBIDO de ('127.0.0.1', 62793)] INIT
[17:54:18] [REGISTADO] Lugar 2 criado.
[17:54:18] [+] Ligação estabelecida com ('127.0.0.1', 62794)
[17:54:18] [RECEBIDO de ('127.0.0.1', 62794)] INIT
[17:54:18] [REGISTADO] Lugar 3 criado.
[17:54:18] [+] Ligação estabelecida com ('127.0.0.1', 62795)
[17:54:18] [RECEBIDO de ('127.0.0.1', 62795)] INIT
[17:54:18] [REGISTADO] Lugar 4 criado.
[17:54:19] [+] Ligação estabelecida com ('127.0.0.1', 62796)
[17:54:19] [RECEBIDO de ('127.0.0.1', 62796)] INIT
[17:54:19] [REGISTADO] Lugar 5 criado.
```

Figura 2 - o terminal do "Parque" confirma a criação de cada lugar.

Protocolo de Comunicação

A troca de informação entre o Parque e os Lugares é realizada através de um protocolo textual desenvolvido no módulo protocolo.py.Todas as mensagens seguem, por definição, o formato geral: <COMANDO>;;param1=valor1;;param2=valor2.
Por exemplo: UPDATE;;id=3;;estado=OCUPADO.

Este formato simples e legível permite codificar e descodificar os dados de forma uniforme em ambos os lados da ligação (cliente e servidor).
O módulo protocolo.py disponibiliza as seguintes funcionalidades:

- codificar(comando, **kwargs) – cria a mensagem formatada para envio;
- descodificar(mensagem) – interpreta e valida as mensagens recebidas;
- Exceções específicas (ComandoInvalido, ParametrosInvalidos, ProtocoloErro) – utilizadas para identificar e tratar erros de protocolo.

Durante a simulação, cada Lugar envia periodicamente comandos UPDATE com o seu estado atual, e o Parque responde com OK (sucesso) ou ERRO (falha), conforme a validação dos parâmetros.
Para testar a robustez e fiabilidade do sistema, cerca de 25 % das mensagens são propositadamente incorretas, de modo a verificar o comportamento do servidor em situações de erro.
Entre os casos simulados encontram-se:

- Formato incorreto: UPDATE;;id → parâmetro mal formado;
- Comando inválido: START → comando não reconhecido;
- ID inexistente: id=9999 → tentativa de atualizar um lugar não registado;
- Estado inválido: estado=INVALIDO → valor não permitido.

As figuras seguintes apresentam um exemplo de interação entre o Parque e os Lugares, com mensagens válidas e inválidas, bem como as respetivas respostas do servidor.

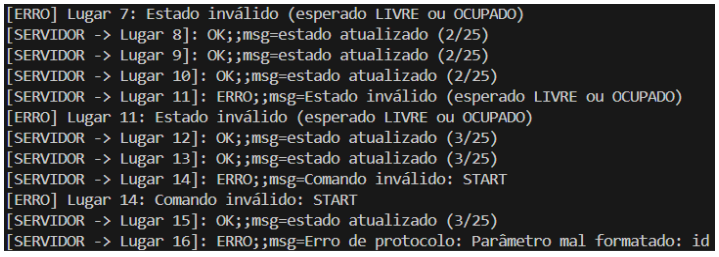

Figura 3 - Mensagens de erro e respostas do Parque.

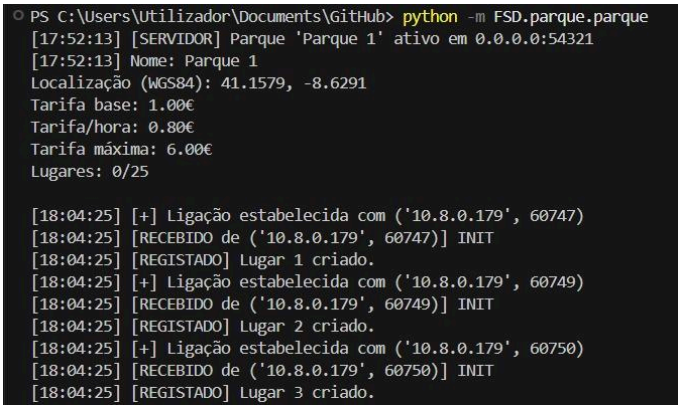

Figura 4 - Atualização de estados entre Lugar e Parque.

Concorrência e sincronização

O sistema recorre a multithreading para simular vários lugares em simultâneo, permitindo que cada thread represente um sensor independente que comunica autonomamente com o Parque.
Esta abordagem assegura paralelismo na comunicação e maior realismo na simulação, uma vez que cada lugar pode alterar o seu estado e enviar mensagens em momentos distintos.

No lado do servidor, as operações críticas — como a atualização do número de lugares ocupados ou a escrita no dicionário partilhado de lugares — são protegidas por um bloqueio (Lock), evitando condições de corrida (race conditions) e garantindo a consistência dos dados partilhados entre threads.
Desta forma, é assegurada a integridade do estado global do Parque, mesmo quando vários clientes enviam atualizações em simultâneo.

Considerações finais

A arquitetura desenvolvida é modular e extensível, com o protocolo.py a centralizar a comunicação, o config.py a definir parâmetros globais e os módulos parque.py e lugar.py a implementar a lógica principal.
O sistema demonstrou funcionamento estável, boa gestão de concorrência e resposta correta a erros. Foi testado em máquinas distintas, com o servidor e os clientes em computadores diferentes, comprovando a comunicação TCP real em rede.
Para permitir esta ligação entre redes distintas, foi utilizada uma VPN (WireGuard), assegurando a conectividade e o encaminhamento correto dos pacotes. A Figura 4 mostra o servidor a receber ligações do endereço interno 10.8.0.179, confirmando a comunicação TCP remota, e a Figura 5 apresenta a interface da VPN ativa durante os testes.

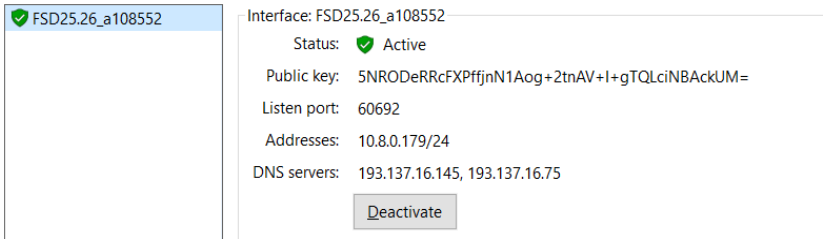

Figura 5 - Interface do WireGuard ativa

A implementação cumpre integralmente os requisitos da Fase 1 e constitui uma base sólida para as fases seguintes do projeto.